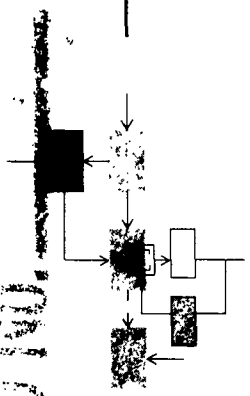
AN ALGORITHMIC THEORY OF LANGUAGE - ROSS - TECHNICAL MEMO [illegible] NOV. 1962 - [illegible]



TECHNICAL MEMORANDUM

Electronic Systems Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE 39, MASSACHUSETT

Department of Electrical Engineering

TECHNICAL MEMORANDUM Copy No. 10

ESL-TM-156

AN ALGORITHMIC THEORY OF LANGUAGE

by

Douglas T. Ross

November, 1962

Contract No. AF-33(600)-42859

The work reported in this document has been made possible through the support and sponsorship extended to the Massachusetts Institute of Technology, Electronic Systems Laboratory, by the Fabrication Branch, Manufacturing Technology Laboratory, Aeronautical Systems Division, United States Air Force under Contract No. AF-33(600)-42859, M.I.T. Project No. DSR 8753. It is published for technical information only and does not necessarily represent recommendations or conclusions of the sponsoring agency.

Approved by: Douglas T. Ross
Douglas T. Ross, Project Engineer
Head, Computer Applications Group

Electronic Systems Laboratory
Department of Electrical Engineering
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

NOTICES

Requests for additional copies by Agencies of the Department of Defense, their contractors, and other Government agencies should be directed to the:

ARMED SERVICES TECHNICAL INFORMATION AGENCY
ARLINGTON HALL STATION
ARLINGTON 12, VIRGINIA

Department of Defense contractors must be established for ASTIA services or have their "need-to-know" certified by the cognizant military agency of their project or contract.

ABSTRACT

The Algorithmic Theory of Language takes the view that processing algorithms define classes of language. A language belongs to a class depending upon whether or not it is properly processed by the corresponding algorithm. Following preliminary statement of n-component element and plex definitions, several General Principles concerning the step-by-step growth of large, complex structures are introduced. The words and symbols of language are then considered to be elements with attractive and repulsive properties which cause them to link together to form linguistic structures. The General Principles are applied to suitable element definitions to yield derivations of successively more elaborate algorithms defining the behavior of these elements, and generating in one left-to-right pass the First-Pass Structure which explicitly exhibits the syntactic and semantic structure of a statement by showing syntactic context by a tree structure and semantic context by the "precedence string". The present development stops with the concepts of major and minor modifiers and leaves ambiguity resolution and other topics to future papers.

This document is a preprint of a paper submitted in November to the Journal of the Association for Computing Machinery for publication in 1963.

TABLE OF CONTENTS

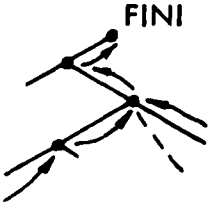
		page
I.	INTRODUCTION	1
	A. Summary	2
	B. Acknowledgements	6
	C. Outline	7
II.	DEFINITIONS AND METHODOLOGY	9
	A. Components and Elements	9
	B. Pointers and Plexes	10
	C. Referents	11
	D. Algorithms	13
	E. Derivation Principles	14
	F. Flow Diagrams	15
	G. Fringe Cuts	18
III.	THE BASIC ALGORITHMIC THEORY	21
	A. The Elements of Language	21
	B. Likes or Attractions	23
	C. The Parsing Algorithm	26
	D. The Fight Algorithm	28
	E. Normal Precedence	32
IV.	SEMANTIC ATTRACTIONS	37
	A. Modifiers	37
	B. The Plike Algorithm	41
	C. Example	43
	D. Decoupling	45
	E. Minor Modifiers	46
	F. Major Modifiers	49
	G. Broken Minor Segments	54
	H. Preliminary Conclusion	55
APPENDIX	ALGOL 60 as an Example	59
BIBLIOGRAPHY		67

List of Figures

Figure 1	Elements of Flow-Diagram Language	page 17
Figure 2	The Parsing Algorithm	27
Figure 3	The Like Algorithm, "f like x?"	28
Figure 4	The Fight Algorithm, "p fight n over x"	30
Figure 5	Example of First-Pass Structure	34
Figure 6	Read/Where for Precedence Algorithm	35
Figure 7	The Precedence Algorithm	39
Figure 8	The Parse-Precedence Algorithm	39
Figure 9	Modifier Modification	40
Figure 10	Modifiers on Minor Precedence	40
Figure 11	The Plike Algorithm	43
Figure 12	Schematic of Precedence when Atom-Atom Occurs	46
Figure 13	Precedence Controls for Minor Modifiers Only	48
Figure 14	Precedence Controls for Major Modifiers	51
Figure 15	Decoupled Parse-Plike Algorithm with Major and Minor Modifiers	53
A-1	Datatypes and Datalikes for ALGOL 60	61
A-2	Left Wordlike Matrix for ALGOL 60	62
A-3	Right Wordlike Matrix for ALGOL 60	62
A-4	The Like Matrix Setting Algorithm	63
A-5	An Example in ALGOL 60 Language	65

ERRATA SHEET

Please note the following typographical errors which affect the accuracy of the text:

Page	
7	Paragraph 2, line 4 . . . are not covered in detail . . .
17	Line 4 from bottom . . . same as A(E).
22	Under Def. 10, line 4 . . . *l*var,
23	Line 5 from top. If i points to a symbol, i → n
28	Fig. 3 . . . the "datatype (x) = X?" yes branch should enter the "wordlike (f) = wordtype (x)?" box.
34	Top of Fig. 5 should look like 
44	Start  (4)
55	Section H, line 3 . . . elaborate behavioral . . .
	Section H, line 4 . . . has been derived . . .

I. INTRODUCTION

In recent years there has been a growing awareness that an important analytic tool in the field of language theory is the concept of a processor for a language. (Gorn, (1), Oettinger, (2), etc.). At the same time that language theorists have been using simple processing algorithms to clarify and rigorize theoretical problems, those most concerned with elaborate processor construction for artificial languages have suffered from a lack of theoretical basis. This paper puts forth the thesis that processing algorithms should be the primary, not a secondary, tool of the language theorist, and that artificial languages should be defined within a sound theoretical framework, in which case sound and efficient translators are an automatic fringe benefit.

Actually the theory presented here is the first major foundation stone of a more general Theory of Plex Processing (Ross, (3)), and, as will be made clear, the importance of the algorithmic approach to language theory is not due to the use of algorithms themselves, but has deeper roots. The contention is that the many complexities and apparent vagaries of language are the natural consequences of certain inherent behavioral properties of words and concepts themselves, and the algorithms which are used are merely the formal descriptions of these "natural laws" which govern the behavior of words and concepts. The meta-theoretic philosophy which is followed here is based on the hypothesis (and it is a hypothesis, open to test) that these natural laws of words and concepts, like the natural laws of matter and energy, obey discoverable minimization and conservation principles, and that the pursuit of these principles, while at the same time applying them to formulate, test, and modify more and more comprehensive versions of the behavioral laws, is a valid and promising approach to this most difficult problem.

The theory of language initiated here represents, then, only the first few small steps toward a complete theory. No claim is made to broad areas of applicability at this time in natural language, but the impact on artificial languages of even the present results is considerable, and further inroads into natural language are clearly indicated. Since the theory is by its very nature evolutionary, we are not concerned directly with known aspects of language which do not at present fit the theory, but with those that do fit the theory, and how well they fit. It is hoped that impartial study will confirm that in the areas which are covered, this nascent theory provides elegant clarification of several vexing but simple aspects of language as we know it, and holds promise of further significant development.

Since the material in this paper relates to language theory, language translation, compiler construction, etc., it is impractical to attempt to sort out and directly reference all of the influences and related work which have been published in these fields. Instead we reference a few of the more closely-related efforts in the context of a general preview of the results which are presented in detail in the body of this paper. Particularly relevant as general references are the entire issues of references 16, 17 and 18. Many of the papers in these special publications speak to problems covered here, though frequently the emphasis differs.

A. Summary

Language is both form and meaning -- syntax and semantics. Any effort to artificially consider only form or syntax without at the same time treating meaning or semantics, or vice versa, decimates the very concept of language, and can at best be only constructing formalisms of structures which have language-like properties. The inextricable nature of form and meaning must be recognized as a prerequisite to considering language as such.

We restrict our initial attention to what we call linear sequential languages, i. e., languages in which words and symbols occur in a sequence, left to right in an input string. We consider each word to be represented by an n-component element whose components show left and right syntactic and semantic context, type, and other features, by means of pointers. In a single left-to-right scan of the input string, an algorithm transforms the input string into the first-pass structure in which all pointers are properly set. The form or syntactic structure of the statement is shown explicitly by the parsed tree structure which results from setting the left and right syntactic pointers, and the meaning or semantic structure is shown explicitly by the linked rings of the precedence string which results from setting the left and right semantic pointers. The meaning may be evaluated by following the precedence string, which shows which words precede other words in the establishment of total meaning, and at each step the proper syntactic context is known. The present theory is restricted to ordinary words, and major and minor modifiers, but later extensions will treat ambiguities, non-trivial context-dependency, and other more elaborate features.

In addition to the algorithms themselves, considerable emphasis is given to the methodology of the theory. The viewpoint is that large, complex structures (such as are found in language) never arise all at once, but are built up step-by-step out of simpler structures. A set of five General Principles are given, the most important being the Immediacy Principle and the Stacking Principle, and these are used to derive algorithms which define the behavioral properties of elements and cause them to link together to form large structures. While the Immediacy Principle, which says that settings should be made immediately as soon as conditions are right, has not been explicitly stated elsewhere, it has been used widely in many of the "one-pass" processors which have been written. Similarly the role of stacks, which

operate on a last-in-first-out basis, has been widely recognized, particularly by Samelson and Bauer (4), and Oettinger (2), although virtually every recent processor also uses a stack.

Oettinger introduces the concept of a " Δ_M Theorem" which proves for a given algorithm using a stack, that a well-formed "middle" of a formula will leave no residue on the stack. This important concept allows a type of induction to prove the validity of algorithms. Sherry (5) proves several such theorems for algorithms based on the "predictive analysis technique" of Rhodes (6), and many of the proofs of Markov (7) embody the same principles independently. While all of the algorithms of this paper are subject to analysis by Δ_M Theorems, the proofs have not actually been carried out because our present terminology and knowledge of technique precludes elegant proofs. For simple algorithms as treated by Oettinger and Sherry, the proofs, although lengthy, are not too unwieldy, but the added complexity of the algorithms of this paper makes straight-forward proofs impractical.

The role of the stack is also different from most algorithms which have been described in the literature. Like many other studies, stacking is triggered by certain relations which apply between "operator pairs". Many algorithms, (including the early precursors to those of the present paper), depend upon a heirarchy or binding strength relation between operator pairs, (Samelson and Bauer (4), Huskey (8), and others). The heirarchy approach is, however, only of limited (though significant) power, and is rigorously covered in an elegant fashion by the "precedence grammars" of Floyd (9). The limitations of the heirarchy approach are apparently overcome, in great measure by the "predictive" techniques of Rhodes (6), Sherry (5), Irons (10), etc., but at considerable expense in efficiency.

All of these approaches appear to be strongly influenced by the concepts of "productions" (Chomsky (11), Gorn (1)) and by the use of meta-linguistic descriptions of grammars using notations such as the Backus Normal Form, Naur (12). The view of this paper is that such meta-linguistic descriptions are tabulations of the natural consequences of basic properties of the words and concepts of a language, and are more the product of the analytic approach, than of the language itself. Many of the meta-linguistic types which are used in these forms of description appear from this viewpoint to be names applied to gross structures whose genesis is at a more fundamental level, and is based upon the "likes" or attractions between words. The only meta-terms which are acceptable or even needed, according to this view, are the basic type classes of the words themselves, as modified according to context by the type computation. At the present stage of development, the meta information is given in the form of "Like Matrices" and "datatype" specifications for the words of the language, but later extensions of the theory will provide further internal structure in this area.

One consequence of the change of emphasis from meta-terms to the words themselves is that the parsing part of the First-Pass Structure does not refer to any meta information, but structures the right and left context of each word directly. Whereas most parsing is done in terms of meta descriptors such as "noun phrase", "predicate", etc. (see e.g., Yngve (13) and others) these terms play no useful role in the present theory. The only reference known to the author which uses a similar parsing (for Algol, etc.) is Warshall (14), in which the importance of parsing to determine context prior to evaluation is also recognized, although not in the form of an explicit step-by-step precedence string.

Finally, the close interconnection of semantics with syntax does not appear to have received wide emphasis in the literature, although every machine translation project has of course considered both aspects to some extent at least. Special attention should be called to the work of Ceccato (15), as being the closest in viewpoint to the present paper, although specific techniques are quite different. By attacking the translation of meaning rather than form, Ceccato's ideas correspond quite closely to those of this paper.

A (recursive) perusal of the references will disclose a great many similarities between aspects of this paper and the work of many others, including some which should, perhaps, have been mentioned here. It is hoped that these omissions will be excused, and that disputatious claims to priority will not becloud the acceptability of the theory presented in the following sections.

B. Acknowledgements

A preliminary and very rough presentation of the ideas of this paper was given in an internal memorandum in July 1962, and was distributed to the ASA X3.4.1 and X3.4.2 Standards Committees, the U. S. members of the IFIP Algol Working Group, and other selected individuals. The helpful comments, criticisms, and suggestions of many of these people have been incorporated into the present paper, which is considerably different in organization and presentation.

Particularly fond and substantive acknowledgement is due to Mr. Jorge E. Rodriguez of the Computer Applications Group, Electronic Systems Laboratory for his invaluable contribution to this work. His timely and well-modulated balance of criticism and enthusiasm, coupled with a sensitive insight and understanding, has in many hours of discussion throughout the evolution of this theory been instrumental in its successful development. His many contributions are most gratefully acknowledged.

C. Outline

The presentation of the Algorithmic Theory of Language separates naturally into several parts. We begin with several sections in which the basic concepts of plex processing are rigorously defined. These definitions and the viewpoint which they engender enable precision of terminology in the theory itself, and in the presentation of algorithms in a simple flow diagram language, which is also informally presented. The concept that complex structures are built up step-by-step out of simpler structures is made concrete in the form of five General Principles which are used as axioms in the derivation of algorithms and as guides in the establishment of appropriate element definitions. The use of these principles in a derivational context compensates somewhat for the omission of complete lemmas, theorems and proofs concerning the validity of the algorithms and the behavior of the classes of language which they define. Finally the concept of fringe cutting, which is a very useful methodological concept for solving complex problems in gradual stages is introduced.

The careful preparation of definitions and methodology then permits the language theory itself to be unfolded in a straightforward manner. Following an informal description of those aspects of language which underlie but are not dovered in detail in the present theory, which take the form of datatype specifications and Like Matrices, and provide the basic mechanism for attractions between words, the principles are applied to derive the first algorithm, the Parsing Algorithm. The Parsing Algorithm is concerned solely with form or syntax, and the basic level of theory is completed by the introduction of normal precedence via the Precedence Algorithm. The syntactic and semantic features of these algorithms are then combined by merging the algorithms into a single Parse-Precedence Algorithm which defines the simplest complete class of languages of the theory.

The semantic question is then considered in more detail by the introduction of the concept of modifiers. Then semantic attraction between words is introduced in the form of precedence likes, or "plikes", which are then applied to both major and minor modifiers to derive the final algorithm of the present paper.

Although the appendix contains a description of the Algol 60 Language in terms of the theory, extensive discussion of the implications of its present state of development are not included, since such discussions can much more profitably be undertaken in the next paper, which will treat the problem of ambiguities.

II. DEFINITIONS AND METHODOLOGY

A. Components and Elements

We accept as primitive certain English words and constructions; and mathematical set theory, logic, and analysis, etc. We also accept certain basic primitives:

Basic object primitive is thing.

Basic structure primitive is has (or with).

Basic action primitive is contain.

Basic reference primitive is name.

The first defined object is a component.

Def. 1 "A generic component has a unique name and a place-holder which may or may not contain contents."

In this definition sentence, the words has, name, and contain are used with the English words A, unique, which, may, or, not, and ., which along with similar words are accepted as primitive (including the word primitive itself!), to show that the concept generic component involves two sub-parts called name and place-holder, and that place-holder is the kind of thing that can contain something. Furthermore, contents is the thing contained in the place-holder of the generic component referred to by name.

Def. 2 "A specific component is an instance of a generic component, but its place-holder may contain different contents. We say the specific component is of the same kind as the generic component."

There may be any number of specific instances of a generic component, and no restrictions are placed on their contents. We refer to generic and specific components by the collective word component when the distinction does not

apply. Thus, in brief, "a component is a named place-holder containing contents". Note that all components with the same name are of the same kind.

Def. 3 "A generic element is a generic component whose contents is a set of components, no two of which are of the same kind. The components in the contents are referred to as the components of the element, and the place-holder (i.e., the thing that contains the components of the element) is called the guts of the element."

Def. 4 "A specific element is an instance of a generic element, i.e., it has the same name, and components with the same names, but the contents of the components may be different."

We shall restrict our considerations to elements with a finite number, n, of components and shall sometimes speak of n-component elements. In view of this restriction, however, we frequently will use the shorter word element for n-component element without confusion.

We refer to the place-holder of a component A of the element B using the notation B_A. A generic component is considered to be an element of one component whose component name, A, is the same as the element name, A. The place-holder of a generic component is then referred to by A_A. A specific component must be contained in some element.

B. Pointers and Plexes

With the concept of element well-defined, we now introduce as another basic primitive, an existence space which will be considered to underlie the entire conceptual framework we are building. Consider the set of all guts of elements, and a set of "sites", of the same cardinality. The existence space is a unique set of ordered pairs associating with each guts a site. We

consider the guts to exist at the sites, and the associated member of the site set is called the site of the guts. Both sites and guts are things, and therefore may be contained in components.

$$\text{Existence space} = \{(\text{site, guts})\}$$

Def. 5 "If a component contains a site, we say the component contains a pointer to the element whose guts are associated with the site in the existence space, i.e., the contents is a pointer."

We assume that letters, the real and complex numbers, the Boolean primitives True and False, and similar things are pre-defined as elements (whose names are themselves), and also that they are included in the site set, and that in the existence space, they constitute their own sites as well. Thus any metric (i.e., measuring) process may be viewed as pointing to an appropriate existing measuring scale.

Def. 6 "A plex is an element containing only elements or pointers. A pure plex contains only pointers."

I.e., a plex cannot properly contain an un-named thing -- it must have an internal, named structure. A pure plex may loosely be interpreted as "an interconnected set of n-component elements".

C. Referents

For notational consistency we now make the referencing of contents of components more specific. The notation A(B), read "A of B" is interpreted as follows:

Def. 7 "If B_A properly contains an element C, or contains a pointer to C, then A(B) is a pointer to C. If K is a pointer to B, then

A(K) is the same as A(B). If B_A contains a non-element primitive thing, then A(B) is the thing itself."

For generic components considered as elements we write A instead of A(A). We now can make more general, and at the same time more specific, the concept of naming parts of plexes by the following definition:

Def. 8 "A thing x is internal to a plex p if there exists a sequence N_i, $1 \leq i \leq k$, where N_1 is the name of a component of p, N_2 is the name of a component of $N_1(p)$, . . . , and where x is contained in or pointed to by $N_k(\ldots(N_2(N_1(p)))\ldots)$. The sequence $N_1, \ldots, N_k$ is called the reference chain of x with respect to p, and $N_k(\ldots(N_2(N_1(p)))\ldots)$ is called the referent of x with respect to p."

Thus anything which is contained in a plex has a (not necessarily unique) referent.

Note that the concept of plex so far is primarily one of structure, i. e., containment either directly or by pointer, but on the other hand, every aspect of that structure is named as well, either directly or by a more or less elaborate referent. As the basic object-primitive thing indicates, literally anything can be considered a plex, providing only that in the process of considering, the structure is exhibited in detail, and named. Concrete things, such as table, chair, etc., must be expressed in plex structure which mirrors their known properties, but even so abstract a thing as an as yet unconceived concept can be concretely represented if it is merely given a name.

The reason for the emphasis on naming, and the reason that names cannot be taken away from the structural side of plexes is that, except for a

basic primitive level, all plexes must be referenced to other plexes to be meaningful. I. e., our "world-view" allows only a limited number of undefinables, (the non-plex primitive things in components which are not plexes), and all "interesting" things are in fact plexes built up out of other plexes. In order to build large plexes out of smaller plexes, there must be a way of referencing any point in their structures. The names and referents provide the means.

D. Algorithms

We now proceed to the consideration of the machinery required to manipulate plexes. Our view will be, in fact, that the actual "meaning" of plexes, and what makes them "interesting" is solely a result of how they are manipulated. These manipulations will be carried out by algorithms which transform plexes from one form to another. It is very profitable to take a quasi-anthropomorphic view of this procedure, and to ascribe the results of the algorithmic manipulations to the plexes themselves. I. e., an algorithm which uses the referents to things in a plex and causes various things to happen defines properties of the plex itself. In other words we think of the plex as having behavioral properties as well as named structure, and the algorithms linked to the plex through referents are the means of definition of these behavioral properties.

Thus we see that the full concept of plex actually consists not only of structure and naming, but algorithms as well. The full interpretation of any component of a plex involves every algorithm which makes any reference, however slight, to that component. Conversely, a non-trivial algorithm without referents is downright inconceivable. Thus the two concepts are inextricable. The full concept of plex may loosely be considered to be a structure of things contained in named components, the meaning of the names

(and thereby of the things and structure) being made implicitly explicit by associated algorithms. The idea that every reference by any algorithm influences a plex corresponds quite closely to the philosophical view of modern physics which is expressed in the uncertainty principle.

E. Derivation Principles

We have now laid all the mechanical ground work for deriving the algorithms of the language theory. In order to provide motivation for the algorithms of the theory, however, and to show that they are not arrived at whimsically, we state some General Principles of deriving algorithms which will be followed throughout this presentation. These principles may be considered, along with the definitions, to constitute a type of axiom set, and in the derivation of the algorithms we shall call upon these principles as concise statements of the methodology employed.

Princ. 1 The Simplicity Principle: "In solving a problem, the simplest set of assumptions should be employed along with a minimum of mechanism."

Princ. 2 The Efficiency Principle: "Every operation should be considered to have an associated cost, and the total problem includes due consideration of this cost. In general, if the addition of some simple static mechanism, such as a pointer, can be used to provide an immediate result which otherwise would be obtained by a dynamic recursive procedure of unlimited extent, the use of such a pointer is considered simplest and most efficient."

The Simplicity and Efficiency Principles are complemented by the Immediacy Principle which is in a sense dependent upon them.

Princ. 3 The Immediacy Principle: "Whenever sufficient information is available for a component to be set, the setting should be performed immediately."

Multiple application of the Immediacy Principle in turn leads to the Stacking Principle.

Princ. 4 The Stacking Principle: "Whenever two components of the same kind require setting, the element containing the older component should be set aside on a stack, (operating on the last-in-first-out principle)."

Finally, the basic methodology is completed by the Minimum Limitation Principle.

Princ. 5 The Minimum Limitation Principle: "Whenever there is a choice between two ways of accomplishing the same task, that way shall be chosen which imposes the least stringent requirements on the pieces of the problem."

These principles mirror not only the behavior which we expect and desire of a language, but also are in harmony with the more general view that large and complex structures do not arise instantly, but are built up step-by-step out of smaller structures. They apply to any process in which the new material is supplied one unit at a time, as in the input string. In subsequent sections we will take this view of many problems which are not directly linguistic in nature, and, will apply the principles in the derivation of algorithms and element definitions.

F. Flow Diagrams

We now give an informal definition of a flow-diagram language which is most natural for plex manipulations. We suspect strongly that algorithms

written in this way span the same "space" as, and are in fact equivalent to, the Normal Algorithms of A. A. Markov, the various algorithm theories based on recursive functions, and Turing machines. In any case, since it is fairly clear that any short-comings of the method of stating algorithms can either be repaired, or the algorithms can be rephrased, (with no inconsiderable effort, however), we proceed to the mode of expression which seems best suited to the plex concept.

We begin by adding to our previous definitions the following assumptions:

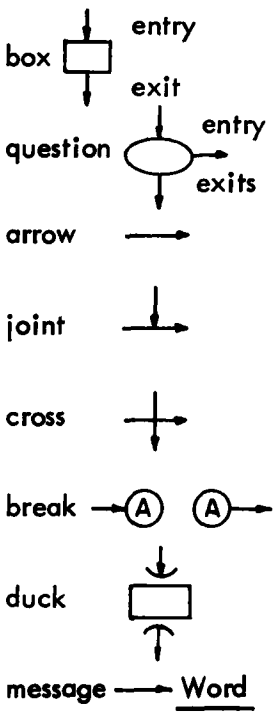
Assumption 1: "Every element contains a unique component named type, whose contents is a code (i. e., a pointer to a reference scale) which specifies what kind of thing the element is."

The concept of a unique type for every element is very basic, but is introduced here in this form as a temporary assumption, rather than as a definition, because its proper treatment is very deep and requires much more machinery than we have at present. The present expedient allows us to refer precisely to type, and will suffice for the moment.

Assumption 2: "We will consider only pure plexes."

This assumption means that we need only be concerned with the manipulation of pointers. Note that there is absolutely no restriction here, since any plex which properly contains a sub-plex (and therefore is not pure) is trivially equivalent to (i. e., may be replaced by) a pure plex which contains a pointer to the sub-plex in its corresponding component.

To save space we introduce the elements of the flow-diagram language graphically and informally as shown in Fig. 1.



Conventions

Only one entry and exit per box.

Only one entry per question.

Flow follows arrows, crosses ignored.

Message STOP ! must occur at least once in an algorithm.

Question may contain only one query.

Box may contain any number of statements, executed from top down.

Any precise language may be used for statements and queries if it doesn't violate rules of flow.

Referents are the "nouns" of the language of statements and queries.

Fig. 1 Elements of Flow-Diagram Language

Clearly more precision and rigor are called for, but we assume these conventions will make the algorithms to follow sufficiently understandable.

The only additional remark that is necessary is to point out that we will make full use of Assumptions 1 and 2.

E. g., Let A, B, be component names

E, F, be element names

α be a mathematical function or operator.

Then $A(E) \longrightarrow B(F)$ Means make the contents of B(F) the same as F(E). A(E) is unchanged.

$F \longrightarrow A(E)$ Implies that F itself is a component, i. e., contains only a pointer, and that pointer is placed in A(E).

$\alpha(A(F)) < 3?$ Implies that $A(F)$ contains a suitable argument for the real valued function α.

With these remarks as guides, the meaning and interpretation of algorithms will be self-evident. In the future the full plex concept of this form of algorithm will be made explicit by defining the elements as plexes and giving algorithms for their complete processing. Note that at that time, the concept algorithm will itself be a plex.

In order to elaborate significantly on the plex and algorithm concepts we must have a better understanding of language. If language can be formalized and made rigorous, it will provide a much more powerful mechanism for further progress than the sterile formalisms of the limited formation rules allowed in existing symbolic notation. An alternate view is, of course, that rigor in language extends the realm of mathematical formalisms. In either case we now proceed to apply the rudimentary plex and algorithm concepts introduced thus far, to the consideration of language.

G. Fringe Cuts

We take one final slight detour to introduce the methodological technical term fringe cutting, since its use will simplify several future discussions. It has been brought out that every problem, viewed in plex form, has many layers of problem within problem within problem. . . . We may view this structure as a many branched tree, where each generation of branches has more and more members, the outermost generation giving, because of its density, the impression of a fringe. In solving a problem it may be expedient to ignore the detail of one or more of the finest layers, even though we know some of the characteristics of those layers and could penetrate deeper into the problem. When we do this we "cut off the fringe" and stop our solution at a coarser level. Note that this cannot be a brutal

amputation -- the coarser view of the problem still must yield a solution. But if a fringe is cut it must be possible to take the finest level and penetrate still further into detail.

Since almost every solution is a fringe cut, (if all possible levels are considered) we will use the term only where it clarifies discussion. We also will attempt to look deep enough in each case to ensure that we only cut a fringe, and do not "cut corners". Cutting corners distorts a problem and usually leads to trouble. Fringe cutting merely postpones deeper penetrations to another day. Normally we only fringe cut when it is clear that future, more powerful machinery or understanding is needed to handle the deeper level elegantly.

III. THE BASIC ALGORITHMIC THEORY

A. The Elements of Language

We begin our consideration of language with a fringe cut by postulating the existence of two plex structures called the vocabulary table and the symbol table. We have said we would restrict our attention to pure plexes, containing only pointers, but the vocabulary and symbol tables, being unspecified in detail, are not pure, but are the actual places of existence of the entities with which we will be concerned. Whenever we refer to a word we actually mean we have a pointer to the vocabulary table entry, which is an element containing all of the needed properties of that word. Similarly when we refer to a symbol we actually have a pointer to an element contained in the symbol table. (A later paper will prescribe the actual table mechanisms in the context of a computer-based system embodying the entire theory being developed here.)

For the present, we ignore all morphological considerations and assume that a speaker of the language, (or a message generator if you prefer), utters a statement in the language by assembling a string of uniquely recognizable and uniquely deconcatenable entities which an unspecified pre-processor uses as names to locate the corresponding words and symbols in the vocabulary and symbol tables. The pre-processor then transforms the statement into the input string, which will be our starting point.

Def. 9 "An input string is a sequence of pointers to words or symbols. Each pointer has a unique predecessor and a unique successor in the sequence, except for the boundary pointers INIF and FINI. The successor to INIF is the first pointer and the predecessor of FINI is the last pointer in the input string."

Note that the input string is a trivial kind of plex, and that, since the pointers can be triggered by anything at all (there is no restriction placed on the names in the tables) the language theory to be developed is a meta-mathematical theory which can be superimposed on anything that takes place in time, as long as the pre-processor can be defined.

The input string by itself is every bit as uninteresting as any statement in a completely foreign language. It may be possible to observe patterns in it, but no meaning or message can be gotten from it. It is our objective to construct algorithms which will transform the input string into a more elaborate plex structure, called the first-pass structure, which will explicitly exhibit the syntactic and semantic structure of a statement. We start with virtually no linguistic properties ascribed to the things pointed to by the input string, and in successive stages we provide more and more properties in the table entries, and thereby derive richer and richer linguistic interpretations of the input string.

The first-pass structure is constructed from first-pass beads.

Def. 10 "A first-pass bead, x, is an element with the following components (in addition to type).

word, w(x), containing a pointer to a vocabulary table entry.

lvar, l(x), containing nil, or a pointer to a symbol table entry, or a pointer to a first-pass bead.

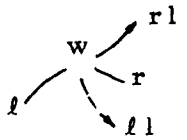
rvar, r(x), containing same as l(x).

minor precedence, l1(x), containing nil, or a pointer to a first-pass bead.

major precedence, r1(x), containing same as l1(x)

A number of stack connectors, s(x), s1(x), s2(x),... for stacking the element, as required.

Such an element is diagrammed as follows:"



We postulate the existence of a "Read/Where Routine" which scans the input string, one pointer at a time, detects whether the pointer, i, points to a word or a symbol table entry, and sets a pointer, n, as follows:

If i points to a symbol, i n

If i points to a word, a first-pass bead, b, is created and

i→(b)

nil→ℓ(b)→r(b)→ℓ1(b)→r1(b)

b→n

Thus in either case, n is a pointer to the "new" thing on the input string. The Read/Where Routine then transfers control to the word or symbol exit.

B. Likes or Attractions

With the input string, the vocabulary and symbol tables, the first-pass bead, and the Read/Where Routine defined, we are now in a position where we can consider the subject of language itself. At this point the only difference between words and symbols is that words have structural properties, (represented by the components of the first-pass bead), whereas symbols do not. As will be developed, ℓvar and rvar have to do with the syntactic structures which can be made by words, whereas the minor and major precedence components are concerned with semantic structure. For the moment we will be concerned only with syntax, and therefore will omit the precedence components from consideration.

It was pointed out that we view elements as having behavioral properties which are described by algorithms which make use of their components. On the basis of these behavioral properties, a given class of elements will construct a particular kind of plex structure in a natural way if they are placed in a

suitable reaction environment, in much the same way that chemical elements combine to form compounds in a chemical process.

The reason that chemical elements will react to form compounds is that certain configurations of atomic nuclei and electrons exert attractive forces on each other, and other configurations repel. Under the proper circumstances, by the making and breaking of chemical bonds, complex molecular structures are built up out of atomic units.

We take this same view of words in a language, (and in fact of all plexes). I.e., every word and symbol is considered to have an underlying structure which causes it to be attracted or repelled, by other words in its environment. Under the proper circumstances words that are attracted to each other will combine to form larger units which in turn have certain attractive properties. Thus the growth of a large syntactic structure out of the atomic units of words and symbols is a natural consequence of the behavioral properties of those words and symbols.

We will not delve into the internal structuring of words and symbols to discover the features which make them attract or repel, but instead will apply a fringe cut and assume that the proper information is available in components of the vocabulary and symbol table entries. Each word and symbol is assumed to have a known datatype. Furthermore, every word is considered to be a binary operator connecting things on its left with things on its right. (In linguistic terms the *l*var component will show the left context of the word and the rvar component will show the right context.) The information concerning the attractive and repulsive properties of the word is condensed into the form of a datalike for both *l*var and rvar, i.e., the datalike left specifies the datatype of thing which the word likes to have in its *l*var, and the datalike right specifies the datatype which the word likes to have in its rvar.

The provision of datatypes and datalikes is not quite adequate to represent the attraction between words without becoming inelegantly cumbersome. Recall that we are cutting off a fringe which undoubtedly contains a great deal of structure to cause words to behave the way they do in a natural fashion, so that it is not surprising that simply specifying datalikes is not sufficient. We wish datatypes and datalikes to be broad and general, so that it is necessary to provide an additional mechanism for accommodating the special cases which do not succumb to broad generalities. We do this by assigning to certain words and symbols the datatype undefined, and then take the perhaps crude (but nonetheless efficient) expedient of listing by case which words like to go with other words, in the form of a binary wordlike matrix.

We assume that the vocabulary entry for each word contains in its wordlike left component a row from the left wordlike matrix, in which each word in the vocabulary table has a column position. A binary one in the i, j entry indicates that word i likes to have word j in its *l*var. Similarly for every word in the vocabulary table the wordlike right component indicates those words which the given word likes to have in its rvar. Since the wordlike matrices are exhaustive, the combination of datatypes and datalikes with wordlikes is an adequate mechanism for the fringe cut. More elaborate and elegant mechanisms are undoubtedly possible, but, since we merely require some mechanism for saying whether or not a given word likes what is on its left or what is on its right in order to know how to fill its *l* var and rvar components, the mechanism described is adequate for our purposes.

We restrict our attention to input strings in which symbol pointers are always separated by at least one word pointer. This corresponds to the assumption that all words are at most binary connectors and simplifies the

derivation. There is no essential restriction in this assumption since any word which should, in fact, act as a multi-argument connector may trivially be replaced by a set of words each of which is a binary connector. For example, a word with three arguments, w(x, y, z), may be considered to be xw_1yw_2z. The present restriction to binary operators can be removed in a future minor elaboration of the theory, if desired.

C. The Parsing Algorithm

We may now consider the Parsing Algorithm which concerns the setting of the *l*var and rvar components in the First-Pass Structure. Since the input string is read from left to right, and *l*var is to show the left context of a word and rvar its right context, by the Immediacy Principle *l*var must be set before rvar. Furthermore the left-to-right scan of the Read-Where Routine plus the Immediacy Principle says that it always must be possible to set the final setting of *l*var immediately. Since rvar depends upon things which have not yet been processed through the Read/Where Routine it may not always be possible to set rvar immediately, so that the Stacking Principle says that there should be a stack of elements whose rvar's are as yet unfilled.

Consider now the input string APBQC where A, B, C are symbols and P, Q are words. As the Read/Where Routine scans the input string from left to right, the word P obtains A as its *l*var and becomes the top-most thing on the stack. The Read/Where Routine continues, reading the symbol B, but since B is a symbol and has no associated first-pass bead the Read/Where Routine will next read the word Q. Now the application of the Immediacy Principle says that if B is the proper rvar setting for P, that setting should be made immediately, or if B is the proper *l*var setting for Q, then that setting should be made immediately. In other words the Immediacy Principle applies simultaneously to both P and Q -- we say that P and Q "fight" over B.

We will assume for the moment the existence of an arbitrator which will decide the winner of the fight on the basis of the datalikes and wordlikes of P and Q. If O wins, then the Stacking Principle says that O goes on the top of the stack covering the word P, since both of them require rvar settings. On the other hand if P wins the fight, then Q must fight with the next thing on the stack, and this fighting continues until, (by the Immediacy Principle), ultimately Q gets its ℓvar set. Then the Read/Where Routine can continue the scan of the input string.

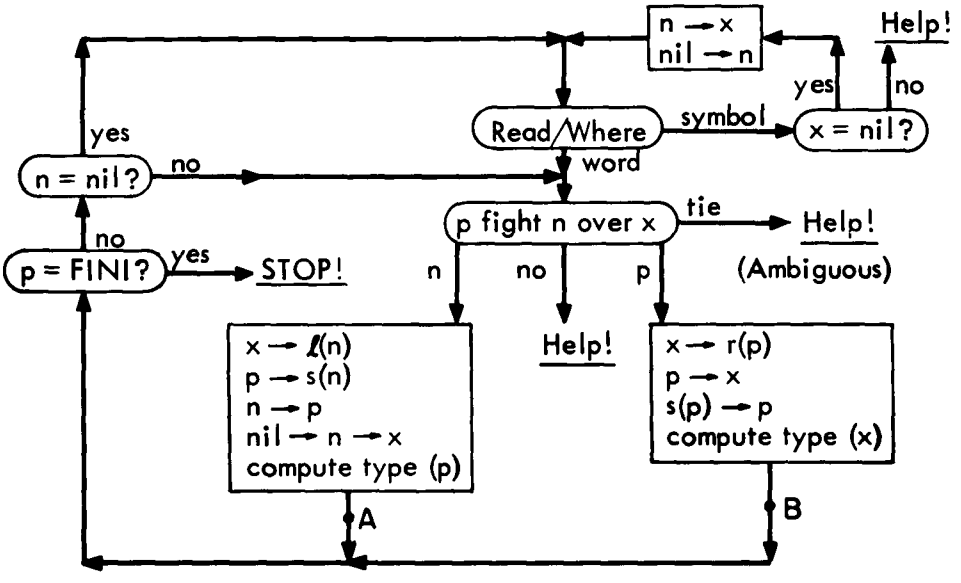


Fig. 2 The Parsing Algorithm

Figure 2 shows the above derivation of the Parsing Algorithm in flow-diagram language. Note that the fight arbitrator calls for help if neither the next word from the input string, (pointed to by n), nor the word on the top of the stack, (pointed to by p), is a winner. The stack and unstack operations are shown by trivial pointer manipulations, and the compute type function sets the appropriate type information into the type component of the first-pass bead. Some words have what is known as dynamic type and

change type value depending upon the type of thing in their *l*var or rvar components. The compute type function is a fringe cut representation of this operation. Note that since the setting of the type component is viewed as setting a pointer to the appropriate measuring scale, the entire algorithm consists of manipulations of pointers.

D. The Fight Algorithm

Before leaving the subject parsing, we consider the fight question in detail. The arbitration of a fight is based upon the likes and dislikes of the words involved. The Like Algorithm, based upon datalikes and wordlikes is shown in Figure 3. Note that the wordlike information is used only when there is insufficient information for the more general datalikes to resolve the question.

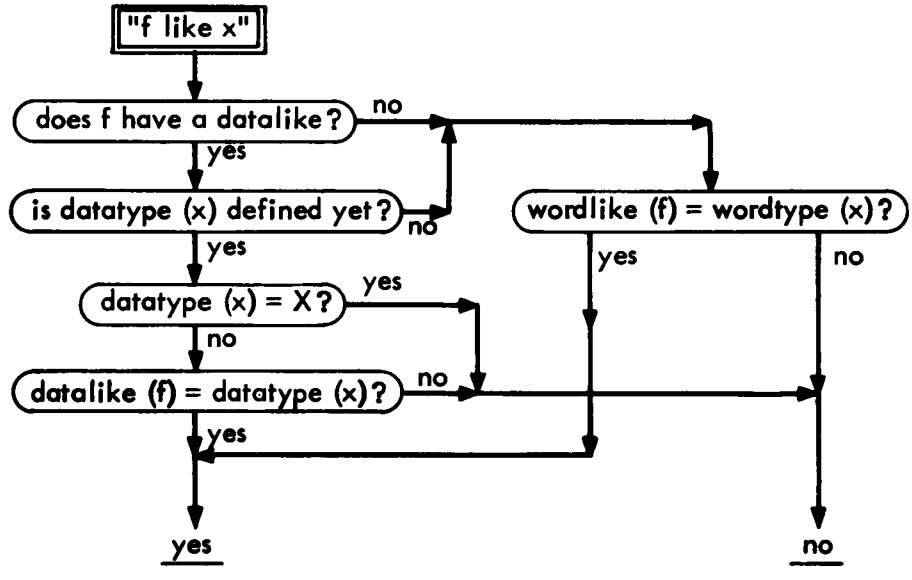


Fig. 3 The Like Algorithm, "f like x?"

In a fight between p and n over x, if neither ℓ(n) nor r(p) likes x, then the Fight Algorithm must ask for help since there is no way of disposing of the unruly x. If either ℓ(n) or r(p) alone likes x, then clearly it is the winner. If both ℓ(n) and r(p) like x, then we examine the consequences of letting either n or p win the fight, by checking the likes of n and p with respect to each other.

If both ℓ(n) likes p and r(p) likes n, then the Fight Algorithm declares a tie, since it has detected an apparently ambiguous situation. The ambiguity problem will be treated in a later paper. In the other three cases of likes between n and p, we tabulate the consequences of letting n or p win, and then call upon the Minimum Limitation Principle to derive the final form of the Fight Algorithm.

Case I. ℓ(n) does like p, but r(p) does not like n.
If n wins then it is required that some other n in the future must like n, since the element beneath it on the p stack does not like it. If p wins then there is no additional requirement, since if the next thing on the p stack loses the fight, n will accept the result.

Case II. ℓ(n) does not like p, but r(p) does like n.
If n wins there are no additional requirements, since even if no future n likes n, the element beneath it on the p stack does like it. If p wins then there is a requirement that some p beneath the present p must like the result, since n does not.

Case III. ℓ(n) does not like p and r(p) does not like n.
If n wins then there is a requirement that some future n must like the present n. If p wins then there is a requirement that some element beneath it on the p stack must like it.

Thus, applying the Principle of Minimum Limitation, Cases I and II give unambiguous results -- in Case I, p should win, and in Case II, n should win. In Case III, the requirement that if p wins some element already beneath it on the stack must like it is more stringent than the condition on n winning, and furthermore, if it is not true for the element immediately beneath p on the stack, the algorithm will break down immediately asking for help. Therefore, in Case III as well, we select n as the winner. Since Cases II and III then give the same result, it is not necessary that the Fight Algorithm check whether r(p) likes n in the case when ℓ(n) does not like p. The final form of the algorithm is shown in Figure 4.

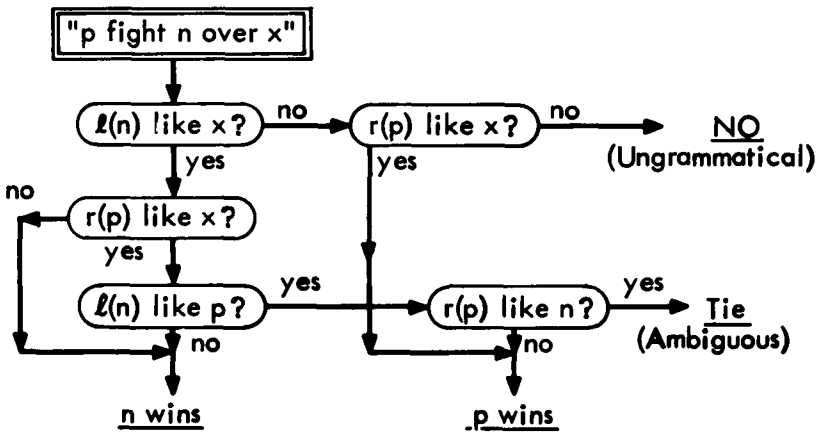


Fig. 4 The Fight Algorithm, "p fight n over x"

The Parsing Algorithm (with the Fight Algorithm inserted) may be considered to be the precise specification of a purely syntactic theory of language in the following sense. The algorithm defines a class of languages in which a given language is a member of the class if, given the vocabulary, the datalike and wordlike matrices can be constructed in such a manner that the Parsing Algorithm correctly parses any statement in the language. Any

language for which these matrices cannot be set up to provide proper parsing for every well-formed statement in the language is not a member of the class. In order to be completely rigorous, we should at this point state and prove a number of lemmas and theorems concerning the status of the various pointers in the algorithm under various conditions, but since this paper is already very long, we will not at this time take the space to present lemmas and theorems whose proofs are trivial if the algorithm is merely followed in detail. In fact it is to be hoped that such theorems can be proved by mechanical proof procedures in the not too distant future and there are indications that this can be done.

It is perhaps worthwhile to introduce at this time yet another viewpoint of the algorithms of this paper which will prove useful in future discussions. The Parsing Algorithm itself can be written as a plex if the flow diagram language elements are defined as n-component elements, and then the entire algorithm may itself be considered an element whose components are the literal pointers which explicitly show in the algorithm, such as x, p and n. It is then very illuminating to think of the algorithm element as being in essence a particle (a la modern physics) and to treat these pointers as state variables so that the various conditions of the algorithm are described by unique states (i.e., sets of values of the state variables). This viewpoint also corresponds of course to that of Turing machines. It is this viewpoint which should be taken in any statement of lemmas and theorems whether they are to be proved manually or mechanically, but again since the Parsing Algorithm is so simple we merely introduce this viewpoint at this time and in future sections will merely describe verbally the requisite facts about the states of the algorithm as they are needed (leaving proofs as an exercise to the reader). Note also that facts about the states of the algorithm

induce similar facts on the statements of any language belonging to the class defined by the algorithm, and may, therefore, be useful for categorizing or understanding languages in terms other than the algorithm itself.

E. Normal Precedence

A language with syntax but no semantics, can hardly be considered a language at all. We now take the first small step into the vast realm of semantics by considering the problem of normal precedence. In order to obtain meaning from a statement it is necessary to evaluate or consider the meanings of the words and symbols used in the statement and the meanings of the structures which they form. Applying the Efficiency Principle, we wish to superimpose on the parsed structure which results from the *l*var and rvar settings of the Parsing Algorithm, an additional structure called the normal precedence string, which will be constructed from pointers showing what words are to precede which other words in the evaluation procedure. I. e., rather than having any scanning procedure to determine meaning we wish to have a direct step-by-step procedure to control the sequence of evaluations.

By the Immediacy Principle, when the input string is scanned from left to right, as soon as any word or subunit can be evaluated it should be evaluated. (Note that here we are applying the General Principles to the specification of a data structure rather than to the derivation of an algorithm, since they are equally applicable.) Since all of the information concerning a symbol is already contained in the symbol table, symbols may be evaluated as soon as they occur, but this trivial evaluation we discard with a fringe cut. Words, on the other hand, are treated as binary operators in parsing, and their meanings depend upon the arguments which are placed in their *l*var and rvar components, i. e., words depend on left and right context. Since a word cannot be evaluated until the meanings of its arguments are known, the

Immediacy Principle says that the arguments of a word must precede it on the precedence string. Since there always must exist at least one word with atomic symbols in both its *l*var and rvar, (this is one of the trivial theorems to be proved), the evaluation process can always be started, and in fact the Immediacy Principle says that it should start with the first such atom-atom word.

The major precedence component, r1var, of a word points to the next word to be evaluated, and we consider that there will be a Precedence String Follower Algorithm which will transfer evaluation control from one word to the next by following the precedence string pointers. If a word has one argument atomic, then as soon as its single non-atomic argument is evaluated it may be evaluated, but if a word has both arguments non-atomic, then (again using the Immediacy Principle) after its *l*var has been evaluated and has sent control to the word itself, the minor precedence component, *l*1var, of the word points to the start of its non-atomic rvar, and only after the entire rvar structure has been evaluated will another major precedence component bring control back to the word, at which time it may actually be evaluated.

Note that the minor precedence component, *l*1var, and the major precedence component, r1var, show the left and right semantic context of a word, just as *l*var and rvar show the left and right syntactic context of a word.

The precedence string is not actually a linear string at all, (although when it is evaluated by the Precedence String Follower the result is the same), but instead has the form of linked rings, each ring containing one *l*1 component and any number of r1 components. Thus it is a complex structure which, (following our viewpoint that complex structures never occur instantaneously but must be assembled step by step), can yield to an input-string-and-algorithm analysis.

Consider the example of a completed first-pass structure shown in Figure 5. The fact that syntax and semantics cannot be separated in a language is shown by the fact that the syntactic parsing structure and the semantic precedence structure are integral components of the complete first-pass structure and cannot be separated. Similarly we wish to derive a single algorithm on the basis of the General Principles, which will construct the entire first-pass structure step by step. Such a combined algorithm itself is a complex structure so that we arrive at it step by step. The completed Parsing Algorithm is the first step, and we now will derive the Precedence Algorithm independently as a second step, and then merge the two algorithms into a single algorithm as a third step. Note that if we consider the writing of the separate algorithms as atomic, and the merging of the algorithms as a "word" binary operator, we are in fact applying the theory which we are developing to its own development -- an entertaining concept which is in fact of considerable significance.

The input string problem for the Parsing Algorithm was handled by a fringe cut which assumed the existence of a suitable preprocessor. In the case of the Precedence Algorithm, however, we can make use of known facts about the Parsing Algorithm (exercises for the reader) to arrive at a suitable input string generator. These facts show (see Figure 5) that

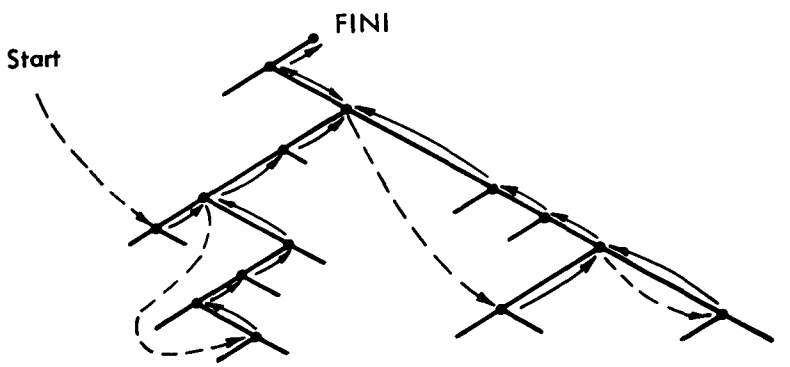


Fig. 5 Example of First-Pass Structure

a major precedence component is to be set whenever a non-atomic ℓvar or rvar is set in the Parsing Algorithm. Similarly, considering the timing of the generation of the parsing structure by the Parsing Algorithm, a minor precedence component is to be set whenever atom-atom occurs, and this can in turn be interpreted to happen whenever an atomic rvar is set into a word whose ℓvar is also atomic. Thus the input string for the Precedence Algorithm consists of the ℓvar and rvar settings of the Parsing Algorithm. These facts give rise to an appropriate Read/Where Routine for the Precedence Algorithm as shown in Figure 6, which obtains its inputs from the Parsing Algorithm.

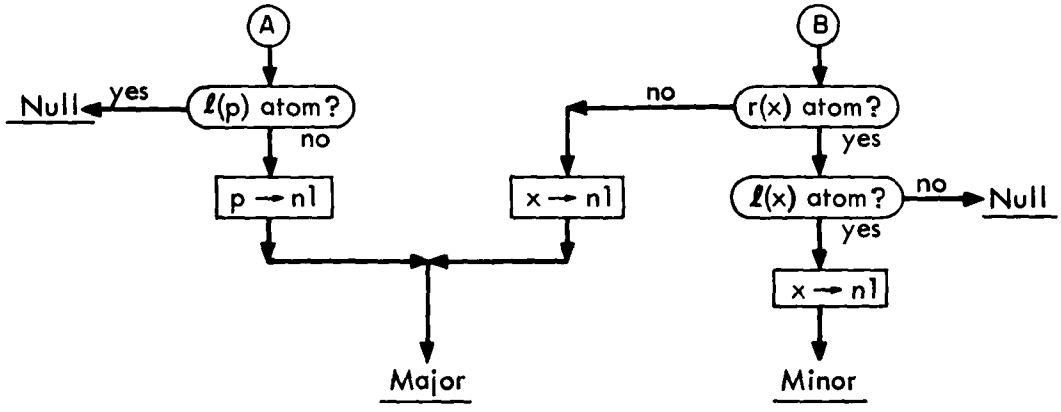


Fig. 6 Read/Where for Precedence Algorithm

The setting of the minor and major precedence components for normal precedence is virtually trivial once the new Read/Where Routine has decided whether a major or a minor component is to be set: The component is set, and then the new "word" becomes the next element whose components are to be set. The process is so simple that we have not even formally called upon the General Principles in this derivation, although they have been applied nonetheless. The Precedence Algorithm is shown in Figure 7, and the Parse-Precedence Algorithm which results from merger of Figures 2, 6 and 7 is shown in Figure 8. Note that in the merger process the Simplicity Principle has been applied to eliminate some steps, but that the separate

functioning of the two algorithms which have been merged can be seen in the combined algorithm. In particular if all symbols with ones are ignored, the algorithm reduces to the Parsing Algorithm.

IV. SEMANTIC ATTRACTIONS

A. Modifiers

It was mentioned that the normal precedence is only the first small step into the realm of semantics. In fact since the normal precedence mirrors so exactly the parsing structure, only the simplest of languages are of the Parse-Precedence class. The class is not empty, however, for it includes standard algebraic notation and many artificial programming languages. Our objective, however, is to go beyond the simple languages into ones of richer expression, and as usual we proceed step-by-step. It was mentioned that the normal precedence string structure has the form of linked rings where the beginning of each ring is shown by a single minor precedence component, but there may be any number of major precedence components in the ring. The asymmetry of the restriction to a single minor precedence component disappears in a natural fashion with the introduction of the concept of minor modifiers. In a later section we will also define a major modifier, but since the distinction between minor and major modifiers can only be made at that time, we now will describe the inclusive general concept of modifier, and our specific remarks will apply to minor modifiers.

We assume that the vocabulary table entry for each word contains two types of information concerning the evaluation of that word, called the minor evaluation and the major evaluation. The minor evaluation information concerns the determination of the meaning of the word when its *l*var has been completed (i. e., when the left context of the word is known), and the major evaluation concerns the determination of the meaning of the word when its total context (both left and right) is known. The normal precedence string discussed above has been concerned only with the major evaluation of the word meanings, and nothing has been said thus far about the minor evaluations.

Consider a word with non-atomic ℓvar and rvar and with normal precedence inserted. There will be a major precedence component from ℓvar to the word, and also a major precedence component from rvar to the word, and the minor and major evaluations are to take place when the Precedence String Follower transfers control along these arrows respectively. Although for many words the minor evaluation is the identity function, some words may take on a certain preliminary meaning based upon their left context, and this minor evaluation may influence the treatment of things in its rvar. Such words are called modifiers because they modify the meaning of words which follow them in the input string. Usually a modifier actually modifies only as much of the input string to its right as is parsed into its rvar, but in any case its minor evaluation function is performed early on the precedence string, before the rvar is complete, and since there is no restriction placed upon the functions performed during this minor evaluation, the modifying power has no essential limitation.

Modifier words with non-atomic ℓvars, ("left context dependent modifiers"), are handled properly by the algorithm of Figure 8, but since the normal precedence string does not visit words with atomic ℓvars, if the minor evaluation function of the modifier is to occur at its proper place in the precedence string, a slight modification to the algorithm of Figure 7 is required. In fact all that is required is to check to see whether a word is a modifier when its ℓvar is atomic, and if so to splice it onto the end of the current ℓlvar which starts the precedence ring. This is accomplished by splicing between the points labelled C and D, in Figure 8, the trivial algorithmic step shown in Figure 9.

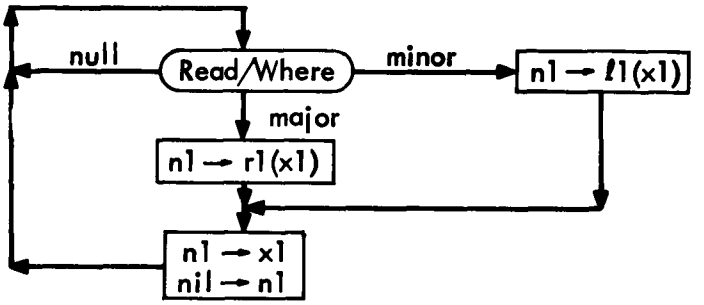


Fig. 7 The Precedence Algorithm

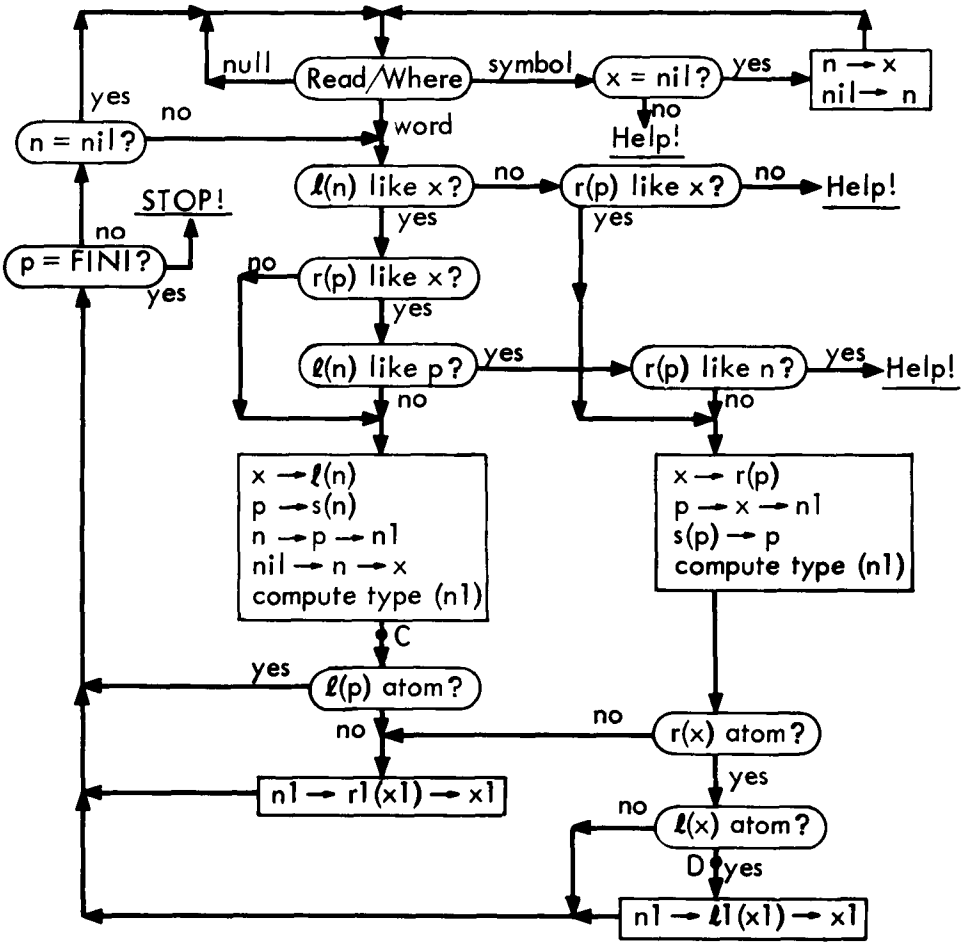


Fig. 8 The Parse-Precedence Algorithm

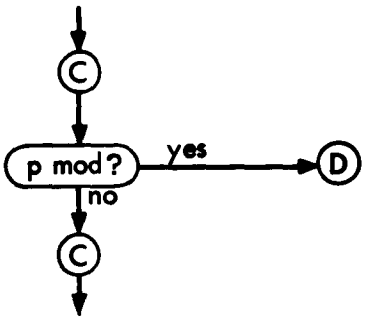


Fig. 9 Modifier Modification

With Figure 9 merged with the algorithm of Figure 8, the resulting First-Pass Structure is now modified so that every minor modifier in an rvar is visited in order by the minor precedence chain, which is terminated by the usual atom-atom start of the normal precedence, at which point the major precedence chain completes the ring, as is shown schematically in Figure 10. Notice that the asymmetry in an individual ring is now eliminated, for it is now possible to have any number of minor precedence components (associated with modifiers) followed by any number of major precedence components.

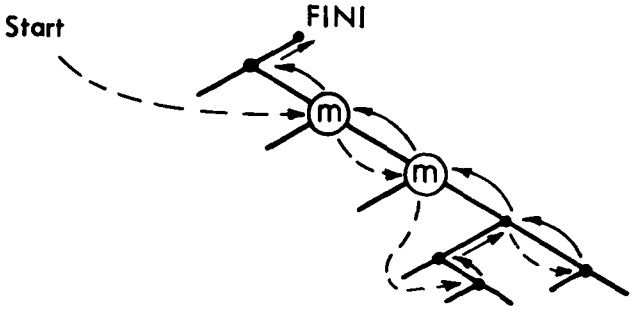


Fig. 10 Modifiers on Minor Precedence

B. The Plike Algorithm

Once again the combined algorithms of Figures 8 and 9 may be interpreted as defining a broader class of languages, and there are many known languages of simple types which are members of that class. Still, however, even with modifiers, we have penetrated only minutely into the realm of semantics, and we now take a more sizeable step. In the preceding sections we have blindly assumed that it was legitimate to let to the precedence components of a first-pass bead point to any other first-pass bead regardless of its type. This is an extremely restrictive assumption and does real violence to our viewpoint of things operating as a chemistry under the influence of attractive and repulsive forces. So we now remove it, and thereby open a veritable cornucopia of further rich linguistic features.

With the Parsing Algorithm as background showing how the General Principles apply in the derivation of algorithms, we experience no difficulty in taking this step. For simplicity of exposition we consider the major and minor segments of an individual precedence ring separately and consider the general problem of linking one-component elements together to make a string, taking cognizance of the fact that each element has a precedence type and each component has a precedence like. We shorten the term precedence-like to the single word plike, and consider the problem of assembling a precedence string segment under the influence of plikes, i.e., whether or not words like to be connected by a precedence component. Just as datatypes and datalikes can be considered to specify whether words like to go together on the basis of syntax or structure, plikes say whether or not words like to go together on the basis of meaning or semantics., i.e., do their meanings match.

As before we approach the problem of capturing the concept of plikes in an algorithm by first deriving from the General Principles the Plike

Algorithm separately, and then we will merge this algorithm with other algorithms, again applying the Principles for simplification.

Since this time we are talking about the setting of only a single component (say rlvar to be explicit) there is no question concerning the applicability of the Stacking Principle -- the Plike Algorithm must include a stack. We call the head of the stack pl. The things to be stacked are precedence string segments, which are "held" by a pointer, xl, pointing to the head of the chain of pointers. With such a segment on the stack, pl, it will fight with the current segment xl, over the new word pointed to by nl. The same Fight Algorithm as was used in the Parsing Algorithm may be used here, using plikes rather than likes, only in this case stacking takes place whenever neither pl nor xl plikes nl. In this case xl is stacked on top of pl, and nl becomes xl, starting a new segment which will fight over the next nl coming in from the input string.

Whenever pl wins the fight, its precedence component is set to point to nl and it is unstacked. The beginning of the resulting segment is then treated as nl and the new pl which is uncovered by the unstacking operation fights with xl over that new nl. As long as pl continues to win, the unstacking continues, and the process terminates when either xl wins, in which case the entire segment becomes xl, or until neither xl nor pl like the nl, in which case the current xl is stacked, and the nl becomes xl, as before.

Since the beginning as well as the end of a segment are of importance here, the Efficiency Algorithm applies and says that rather than searching for the beginning of a segment, a pointer from the head of the segment to the tail should be established so that the segment is made into a loop. We call this operation "wrapping up the segment" so that the things which are stacked are actually wrapped segments. The unstacking operation is then followed by unwrapping the segment and establishing a pointer to the beginning of the segment.

C. Example

The Plike Algorithm is shown in Figure 11, but the way in which a string is re-ordered on the basis of plikes is best illustrated by a simple example.

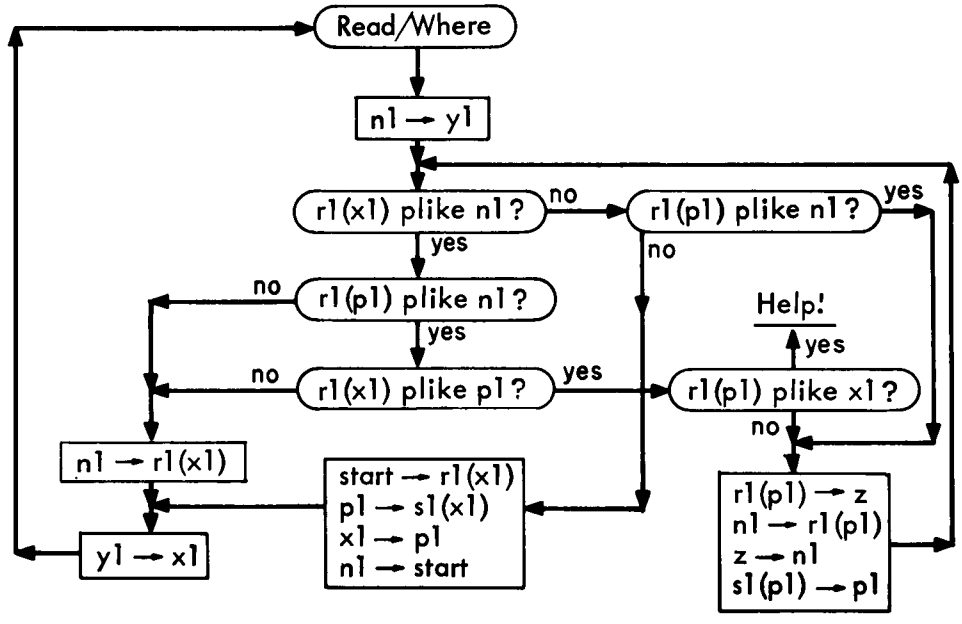


Fig. 11 The Plike Algorithm

Consider an input string of operators from the major precedence setting of the Precedence Algorithm.

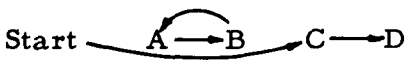
$$A\ B\ C\ D\ E \tag{2}$$

Assume that A plikes B. Then the major precedence component A is set to point B and AB is an established segment of the precedence string.

$$\text{Start} \longrightarrow A \longrightarrow B \tag{3}$$

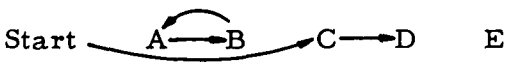
When C is read in, if B does not plike C, then we set the segment ending at B aside on the stack. To do this we "wrap up" the segment by setting the major precedence component of B to point to A temporarily, and set aside the

wrapped-up segment on the stack. C then is made the beginning of a new current segment. When D is read in the configuration is as follows:



(4)

Now B and C fight over D on the basis of plikes, since either B or C could precede D. Assume that C wins the fight on the basis of the Plike Matrix, then when E is read in the situation is as follows:

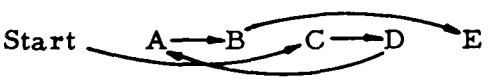


(5)

Then B fights D over E. Assume that in this case B wins, and is taken off the stack and unwrapped to show that A B E is an established segment of the precedence string now.

(6)

Then the empty stack fights D over the beginning of the segment, which is now A. Since empty cannot win, if D plikes A, we have the final precedence string, re-ordered on the basis of plikes, as follows:



(7)

The important thing to notice about this example is that the start of the segment can not absolutely be determined until the entire segment is complete. Notice how the start pointer moves from A to C when the segment AB is stacked. If the segment had been longer and the fights were resolved differently, it would be perfectly possible for the start of the segment to switch back to A, or even switch to some other word farther down the string. This fact has important consequences with respect to the use of the Plike Algorithm for major and minor precedence segments.

D. Decoupling

A precedence ring begins with a minor segment, consisting of at least the minor precedence component of normal precedence, but which may have any number of modifier minor precedences. With the occurrence of an atom-atom the major precedence segment, consisting of any number of major precedence components, begins, and the ring closes when the end of a major precedence component coincides with the beginning of the initial minor precedence component. Thus a precedence ring has a beginning, an atom-atom in the middle, and an end. The minor and major segments of the precedence ring may be reordered independently on the basis of plikes, and at any time in the major segment, the occurrence of a minor precedence component may initiate another precedence ring.

Whenever a new precedence ring is initiated, a new start pointer becomes active and is trying to be set. As the preceding section shows, the start of a precedence segment can not be set until the segment is complete, and since the old precedence ring was interrupted by the initiation of the new precedence ring, its start is also still active. Therefore, the Stacking Principle applies and says that since there are two active start components, the "element" containing the older start component must be set aside on a stack. We call this "decoupling" the precedence rings.

The element which must be stacked is the entire precedence ring itself, and consists of the "state variable" pointers of the Plike Algorithm, namely nl, xl, pl, and start. The current settings of these state variables must be saved or wrapped up in a suitable manner on the stack, so that when the new precedence ring is completed, the generation of the old may continue. We do this wrapping and stacking operation not by constructing an actual n-component element for the ring itself, but by taking the more efficient constructive

approach of using the components of the existing first-pass beads to perform the stacking function. Figure 12 shows the situation schematically.

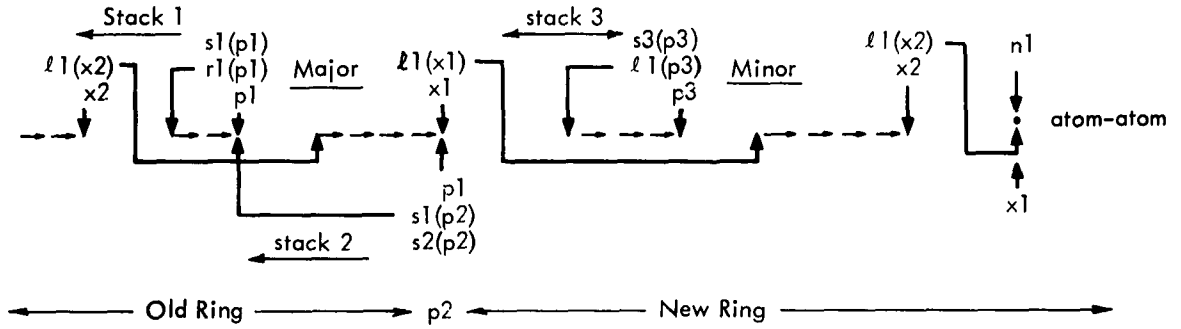


Fig. 12 Schematic of Precedence when Atom-Atom Occurs

Figure 12 shows the interrupted major segment of the old ring including the top-most wrapped-up segment on the p1 stack, and the current major segment whose start is indicated by the minor precedence component of x2, which is the last element in its minor segment. The figure also indicates the minor segment of the new ring, including the top-most wrapped-up segment on its stack, the p3 stack, and the current minor segment whose start is indicated by the minor precedence component of the last element, x1, in the major segment of the old ring. In n1 is the atom-atom element which signifies unequivocally that a new ring is being generated. Notice that since the minor segment of the new ring might consist only of the single minor precedence component of normal precedence, which is triggered by the occurrence of atom-atom itself, the use of the occurrence of atom-atom is required to indicate all rings whether or not they contain modifiers.

E. Minor Modifiers

When modifiers were introduced it was pointed out that our initial remarks would refer to only minor modifiers, and at this time we can now be

explicit about the adjective "minor". A minor modifier is one which enters into the competition in the minor segment of a precedence ring, but if it is unsuccessful in the fights and ends up on the p3 stack, then once the minor segment is terminated by the occurrence of atom-atom, the stacked minor modifier is discarded and not considered further. In other words the minor evaluation function of a minor modifier is taken into account only if it is used in suitable context -- the occurrence of a suitable context being indicated by the fact that the words with which it is associated plike properly so that it becomes incorporated into an active minor precedence segment. If the context of the minor modifier word consists of words which do not plike to be connected to it, then once the minor segment is terminated by the occurrence of atom-atom, the minor evaluation function of the modifier is ignored completely and never carried out, and only the major evaluation function for that modifier word will be carried out when it is reached on the major precedence segment. As we shall see in the next section, major modifiers do not suffer from this drastic treatment.

With the role of minor modifiers clarified, we now may proceed to the consideration of the stacking of the old precedence ring when the new ring is triggered by the occurrence of atom-atom. As shown in Figure 12, the "state variable" pointers p1, x1, and n1 are saved in s1(p2), p1, x1 respectively. The previous ring is stacked in s2(p2), and the current old ring which is on the top of stack 2 is pointed to by p2. These operations are shown in the first three lines of the Atom-Atom portion of Figure 13. The fourth line shows in addition that the start of the major segment of the new ring is set into the minor precedence component of y1, and the reason is made clear by consideration of Figure 12. Recall that the minor segment of the new ring may not exist, in which case the start of the major segment of the new ring would come from x1 of the old ring rather than x2 of the new ring. As will

be seen when we finally collect all of the sub-algorithms which we are now describing into one merged algorithm, y1 will have the value of either the old x1 or the new x2, so that it alone indicates the universally correct element whose minor precedence component can be used as the start pointer for the major precedence segment of the new ring.

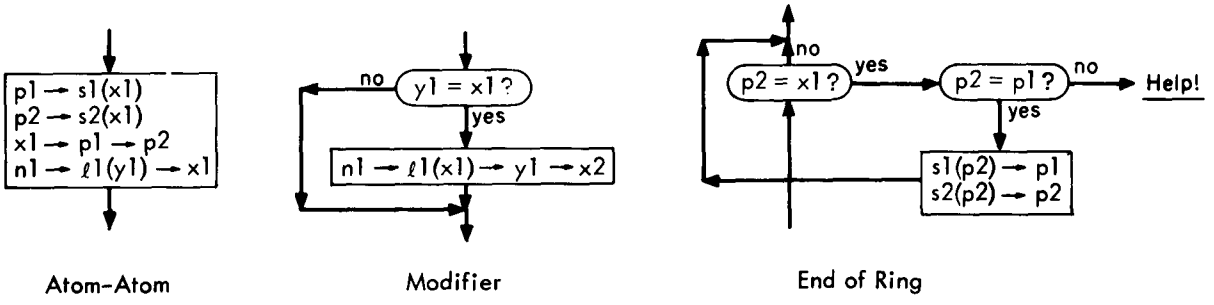


Fig. 13 Precedence Controls for Minor Modifiers Only

The Modifier portion of Figure 13 shows in flow diagram language the initiation of the minor segment of the new ring. Whenever a modifier is detected (see Figure 9) the question "y1 = x1?" tells whether or not this is the first minor modifier in the minor segment. Because the same pointer, y1, is used in both instances of the Plike Algorithm as applied to both the major and minor precedence segments, if y1 is equal to x1 (the current end of the major precedence segment) then that means that the immediately preceding element had its major precedence component set, so that the current element is the first modifier to occur in the new ring. As Figure 13 shows, in this case the new element n1 is set to be the start of the minor segment, indicated by $\ell 1(x1)$, and also the current end of the minor segment, indicated by x2, and y1 is brought up to date.

Our consideration of the precedence controls for ring stacking finishes by the consideration of the End of Ring portion of Figure 13. The closure of a precedence ring is indicated by the fact that the end of the current major precedence segment coincides with the beginning of the ring. Consideration of Figure 12 shows that the beginning of the ring has been saved in p2 when the ring was detected by the occurrence of atom-atom, so that the question "p2 = x1 ?" indicates ring closure. As Figure 13 shows we then check to see whether anything is left on the p1 stack, which contains the wrapped-up unsuccessful major precedence segments. If p2 is not equal to p1, then there are indeed segments on the stack, and the algorithm then calls for help because this indicates that a meaningless statement has occurred. In other words, the checking of plikes in the competition in the major precedence segment shows whether or not the words which are used in a statement go together meaningfully. It is perfectly possible for a statement to be parsed correctly and still be meaningless. This is what is detected at this point in the algorithm.

If on the other hand, p2 is equal to p1, indicating that there are no dregs left on the p1 stack, then the processing of the old ring is enabled by destacking p1 and p2 so that processing can continue. Note that at this point x1 already has the proper value, and the third state variable, n1, will be set by the next operation of the Parsing Algorithm. Recall also that since we are at present considering only minor modifiers, the p3 stack has no role whatsoever in the end of ring operations.

F. Major Modifiers

At this point in the discussion, if space were not at a premium, we would take time out to merge the algorithms of Figures 8, 9, 11 and 13, and present some discussion of the newly defined class of languages covered by that algorithm. Instead, with due apologies to the reader for the confusion

which may result, we now press on to the consideration of major modifiers before the final merger of all algorithms necessary for the complete treatment of that portion of semantics which is covered by precedence and plikes. Recall that a minor modifier was discarded if it was unsuccessful in the competition during the construction of the minor segment of a precedence ring. Major modifiers exhibit more stamina and stay on to fight future battles, as we now discuss.

If a major modifier is frustrated in its attempt to find a suitable semantic context during the generation of the minor precedence segment, so that it ends up on the p3 stack, it then seeks to find a suitable context in any plike fight in the future, whether it is based on either major or minor precedence components, and this aggressive behavior continues through all nestings of precedence rings within precedence rings until finally the major segment generation returns to the modifier. Only when this closure takes place does the activity of the major modifier cease, and if it has been unsuccessful through all battles up to that point, it is summarily discarded just as ignominiously as a minor modifier, so that only its major evaluation has any semantic effect.

As usual, the behavior of major modifier word elements is the result of algorithms which refer to their components. In this case it is necessary merely to provide a mechanism for the p3 stack to be carried along instead of being discarded as in the case where we were concerned only with minor modifiers. We do this by splicing the p3 stack onto the head of a new stack, the p4 stack, whenever atom-atom occurs. We must use a new stack since the major modifiers on the p4 stack are to join in both major and minor precedence fights, and a minor fight will generate its own local p3 stack. The p4 stack then will be a single stack which results from the concatanation of all leftover p3 stacks. Figure 14 shows how this is accomplished.

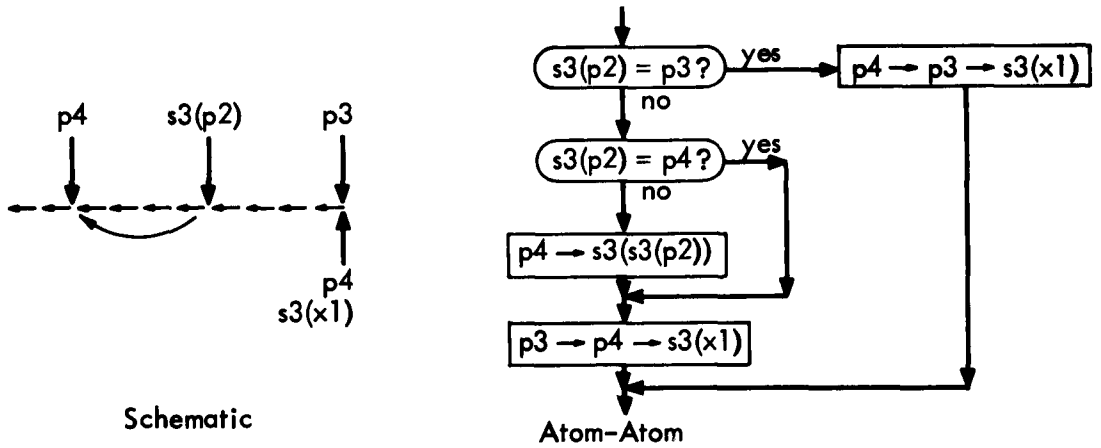


Fig. 14 Precedence Controls for Major Modifiers

The component s3(p2) shows the beginning of the current p3 stack, so that if s3(p2) is not equal to p3, then the current p3 stack contains wrapped-up unsuccessful minor segments which are to be transferred to the p4 stack. Since s3(p2) also indicates the head of the p4 stack at the previous atom-atom, if s3(p2) is not equal to p4, then that means that some unknown number of modifiers have been destacked from p4 so that in order to add the segments on the current p3 stack to p4 it is necessary to make the s3 component of s3(p2) point to the present p4. With this accomplished, the p4 and p3 stacks now constitute a single stack connected by s3 components, so that the p4 pointer may now be moved to the head of the new p3 stack to make the new setting of p4. The new p4 is also placed in s3(x1), since the next step is to make x1 become the new p2, and thus this properly sets the new s3(p2) to coincide with the new p3 and p4. This completes the description of the atom-atom situation when there is something on the p3 stack.

In the other case, when the p3 stack is empty, as is indicated by the fact that s3(p2) does equal p3, then it is necessary merely to set both p3 and s3(x1) to be the same as the current p4, since if p4 has not changed no harm is done, whereas if p4 has changed, this will provide the proper updating.

Thus in either case, when atom-atom is encountered, anything on the p3 stack is added to the head of the p4 stack, so that any major modifiers are saved for future battles.

For the completion of the major modifier discussion we turn to Figure 15, which is the result of merging the algorithms of Figures 8, 9, 11, 13, and 14, with some additional small changes relating to major modifiers and minor segments. Examination of Figure 15 shows that the major and minor precedence settings of Figure 8 for normal precedence have been blown up to include a complete Plike Algorithm in each case. Detectors to test whether the p1 and p3 stacks are in use have also been added, so that the elaborate fight is omitted when the stacks are not in use. In each case, between the box which transfers n1 to y1 and the test for the status of the stack in question, a small algorithm consisting of several questions and boxes has been added. Those on the right-hand side, in the minor precedence portion, consist of a test to see whether or not the p4 stack is in use, and if it is, then if the element currently on the head of the stack is not a major modifier, it is destacked, whereas if it is a major modifier, then if it plikes n1 it immediately wins n1 and is destacked. This shows how the major modifiers on the p4 stack obtain highest priority in the plike fight as they should.

The similar section for major modifiers on the left of Figure 15 contains a question and a box concerned with x2, and also one additional question, following "INIF = p4?", asking whether n1 is equal to p4, when p4 is a major modifier. If n1 does equal p4, then this indicates that the major precedence segment has reached the still unsuccessful major modifier, and it is at this time that it is summarily discarded by being destacked from p4. Otherwise the treatment of the major modifiers on the p4 stack is the same for the major precedence segment as for the minor, i.e., major modifiers have top priority.

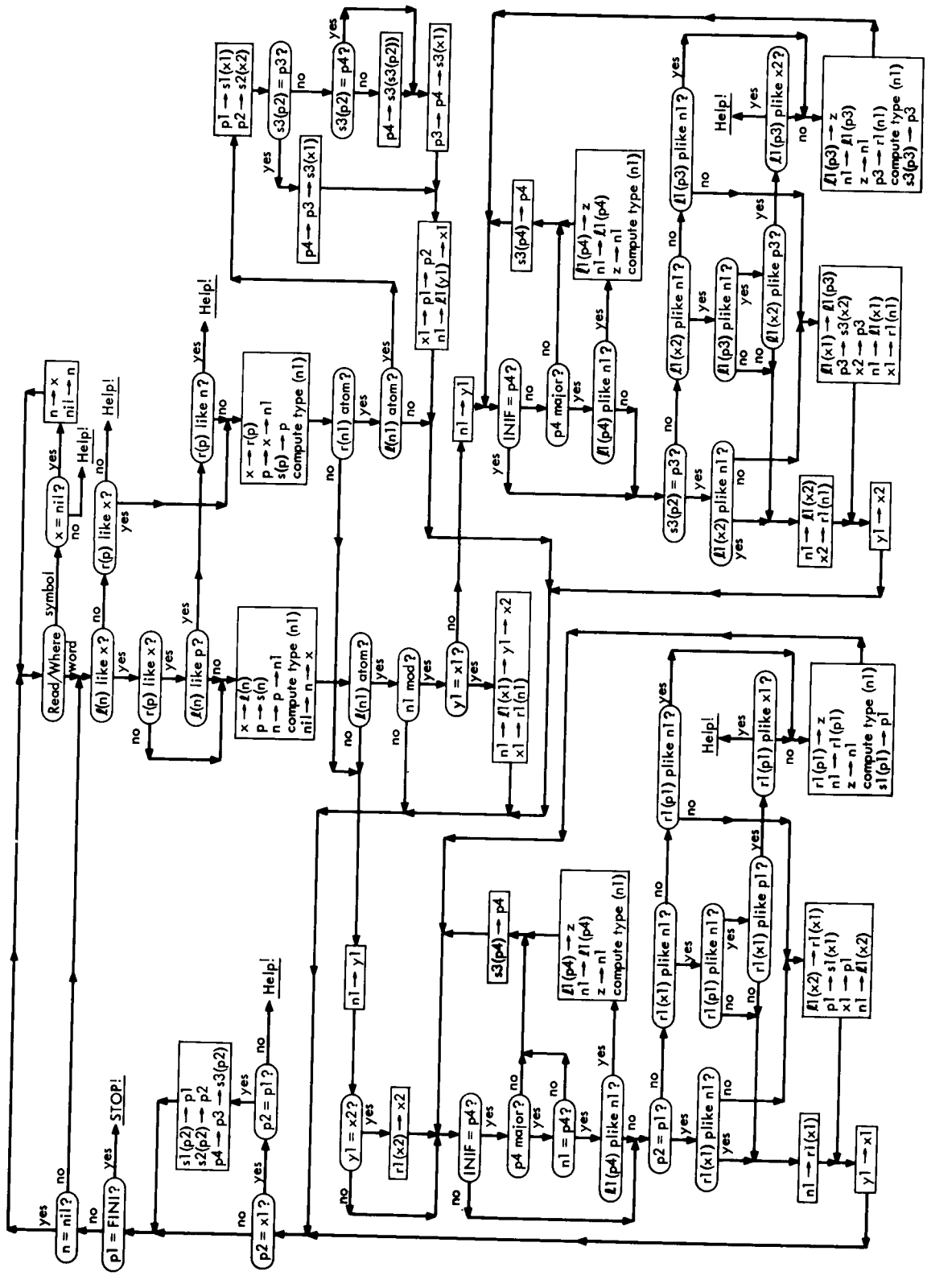


Fig. 15 Decoupled Parse-Plike Algorithm with Major and Minor Modifiers

The only further addition to Figure 15 related to major modifiers concerns the place where the closure of a precedence ring is detected and the p1 and p2 stacks are re-established. One line has been added to update p3 and s3(p2) to coincide with the current value of p4 so that the p3 and p4 stacks are also kept up to date when precedence rings are completed.

G. Broken Minor Segments

The changes concerning the minor segment are somewhat more subtle. Up to this point we have described the precedence ring as having any number of minor components, an atom-atom, and then any number of major components. This mode of description provides proper motivation for plikes in the minor segment, but actually is correct only during the growing phase of the construction of precedence rings. Examination of Figure 10 shows that a precedence ring with many components in its minor segment can equally well be considered to be several trivial rings linked together. Each time a major component reaches a modifier with a minor component, another ring is closed off and serves the same role as an atom-atom for the next higher ring.

When a ring is closed off, it makes a complete unit, and since it serves as the atom-atom for the next higher ring, initiating the major segment, if there is any plike reordering, it may be p1 stacked like an atom-atom whenever a new start is called for. In other words, happenings in the construction of the major segment may cause the minor segment to be broken up and distributed along the major segment as separate sub-rings. In order for this to happen, whenever a ring is closed off, the minor component pointing to it must become the new start pointer for the major segment.

Several small changes have been made in the minor version of the Plike Algorithm of Figure 15 to enable the changing of start as successive rings close off. Whenever a minor precedence component, $\ell 1$, is set, the corresponding major component, r1, is set in the opposite direction, so that

in effect the minor segment as originally laid down is made of two-way pointers. Then, in the major version of the Plike Algorithm of Figure 15, the question "y1 = x2?" detects when a ring closes, and if it does, the component r1(x2) provides the new setting for x2, whose minor component, ℓ1(x2), then serves as the new start.

With major modifiers and broken minor segments, we see that the final precedence string consists of any combination of minor and major precedence components, the precise sequence being determined entirely by plikes. The Precedence String Follower Algorithm will always take the minor component if there is one, unless it arrived at the word from its rvar, in which case it takes the major component. The "p2 = p1?" check that the p1 stack is always emptied guarantees that any word with non-atomic rvar will be so entered.

H. Preliminary Conclusion

Even a cursory examination of Figure 15 shows that the algorithm has now grown to be extremely complex -- a consequence of the rather elaborage behavioral properties of the class of languages which it defines. Were it not for the fact that it has been deriged step-by-step by merging simple algorithms, it would be virtually impossible to give a coherent description of its operation. In view of this complexity it is no longer legitimate to leave the proving of facts about the algorithm as exercises for the reader, but since the present paper is already so long, we defer detailed discussion of the algorithm to a future paper, and instead close with a few general remarks and indications of future developments.

A major deterent to delving deeply into the machinations of the algorithm of Figure 15 at this time is the fact that there are no simple languages known which exercise all of its features. In the Appendix is presented

a brief description of the Algol 60 Language showing the Like Matrices and types, but even that language contains only the most trivial examples of the features which have been discussed. Although an exhaustive examination has not been made, it appears that all other existing programming languages are even simpler (from the point of view of this theory) than Algol 60, but this does not mean that the class of languages defined by the algorithm of Figure 15 is empty. All of the features of the algorithm can be found in suitable selections from natural language, but since well-defined sub-segments of natural language (i. e., English) have not already been defined, elaborate discussions at this time would be out of place.

The very thing that makes natural English too variable to provide a good discussing medium at this time does show explicitly in the present algorithm and will be the subject of the next paper in this series. There are five places in the algorithm of Figure 15 which say help -- one for syntactic ambiguity, two for semantic ambiguity, and one each for ungrammaticalness and meaninglessness. The general principles which have been applied in deriving the present algorithms can also be applied to the derivation of further, still more elaborate algorithms which will tie onto these places so that apparent syntactic and semantic ambiguities are resolved temporarily in a manner which satisfies well-defined minimization properties, and then if on the basis of those resolutions the algorithm runs into ungrammatical or meaninglessness difficulties, then alternate choices of the ambiguous portions are constructed. In the more elaborate algorithms which resolve ambiguities, the algorithm will call for help only when all possibilities arising from all ambiguous situations have been determined unsatisfactory.

Major portions of this extension of the theory have already been derived and await only formal presentation. Once the algorithm has reached that stage, then it will accommodate quite easily a sizeable portion of natural English,

and at the time deeper discussion of the implications of the theory can be made in a natural setting.

APPENDIX: ALGOL 60 as an Example

We show the Algorithmic Theory of Language in action by considering the Algol 60 Language exactly as it is defined in the definitive report of Naur, et al, (12) with the following trivial exceptions:

1. Identifiers, strings, number forms, etc. are assumed handled by the preprocessor and the Read/Where Routine.
2. Square brackets are omitted since parentheses suffice.
3. Since the word procedure is used in two distinct ways:
   (a) the alternate form
   "define (x) ... where ... to be ..."
   is used for
   "(y) procedure ... ; ... ; ..."
   where x is real, boolean, integer, procedure
   and y is real, boolean, integer, empty
   (b) the word procedure is used only as a declarator.
4. Since : is used in three distinct ways:
   (a) the form
   "array (A to B)" is used for "array [A : B]"
   (b) the form
   "...) L:(..." is omitted for the moment.
   (c) the character : always signifies labelling.
5. Since "if ... then ... ; ..." seems only to add confusion and inconsistency, else must always be used.

Figure A1 shows the datatype and datalike information for the full vocabulary. The types dr and dℓ take on the value of whatever is actually parsed into the rvar and ℓvar, respectively. The modifier column shows a fringe cut treatment of context dependency in that the occurrence of a for causes := to be a major modifier. We use the notation for (:=) to indicate := in the context of for.

Whenever the undefined type, x, appears, then the Left or Right Wordlike Matrices, shown in Figures A2 and A3, must be checked. A one in the intersection of row i with column j shows that word i does like word j in its ℓvar or rvar; a zero indicates i does not like j; and a blank shows a don't care case which cannot arise.

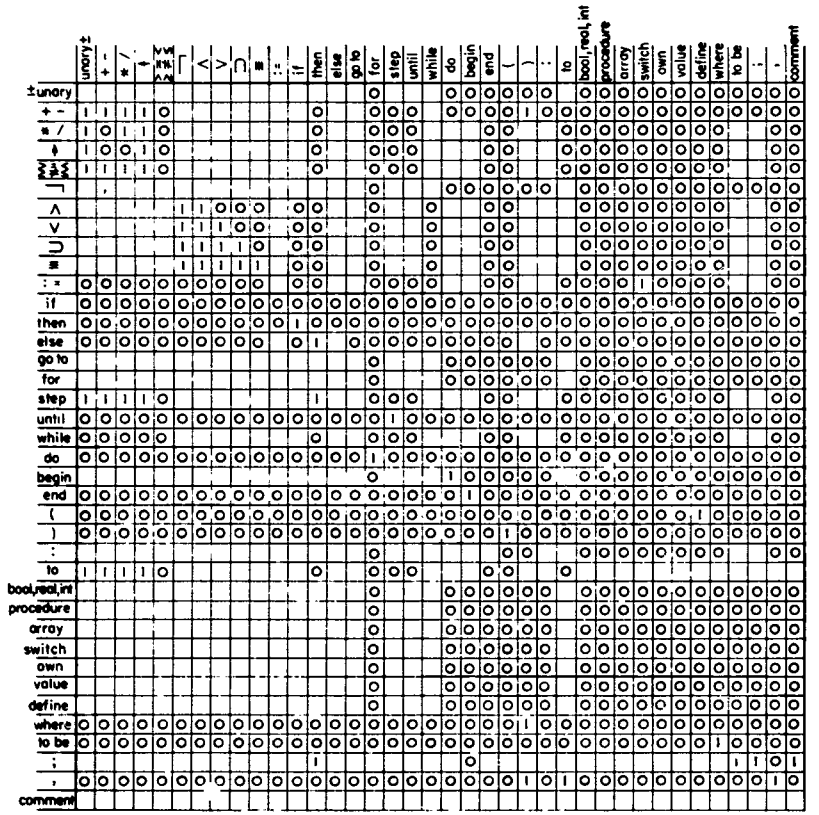
A highly useful adjunct to the basic algorithms of the Theory is the algorithm of Figure A4, which is used to set the proper entries into the Wordlike Matrices. Consideration of the Like and Fight Algorithms, (Figs. 3 and 4), allows Figure A4 to be derived to separate out those word-pairs which are not uniquely determined by the type specifications (e.g., Figure A1, for Algol 60), and which may whimsically be selected by the human. With this algorithm it is an easy matter to set up a selected language according to the theory.

Even though the datatypes and datalikes of Figure A1 have purposely been restricted over what would seem most natural, in order to correspond as closely as possible to the official Algol 60 Report, it is interesting to note that certain generalizations (which actually correspond to minor inconsistencies in the official description) result from application of the theory. One example will suffice to illustrate the point: In order to permit "A := B := C := ...", the := includes the datatype dr, but once this is done, then expressions such as "A := B + C := D * E", which are excluded from the Official Report, are both grammatical and meaningful. They could only be excluded by artificial means. A later paper will give full natural datatypes

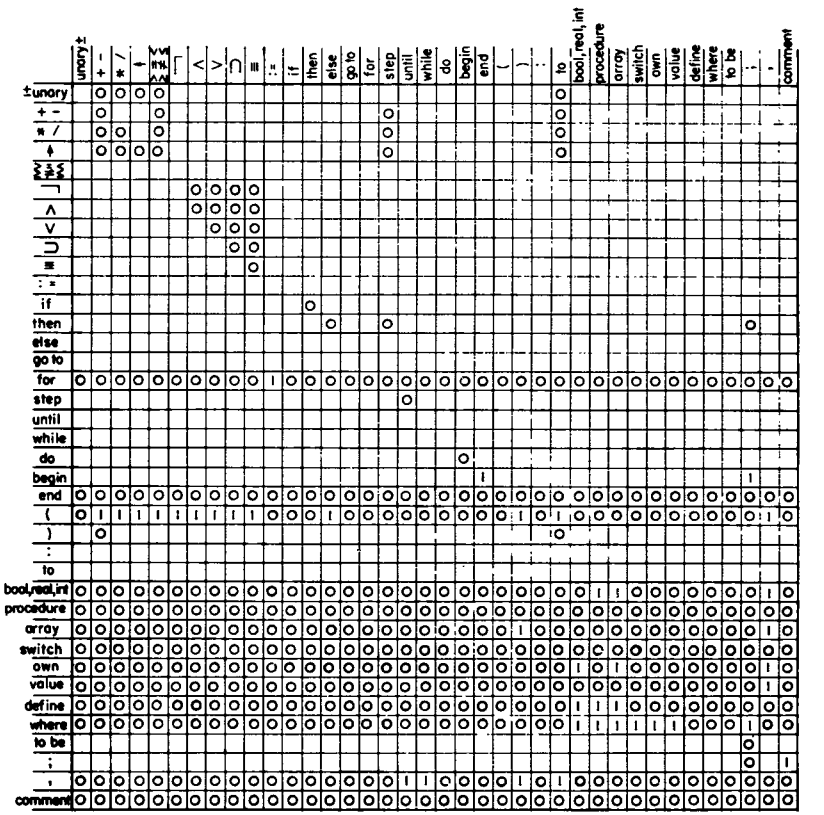
Word	Data Type	Data like Left	Data like Right	Modifier
unary + -	R	E	R	
+ -	R	RvE	R	
* /	R	R	R	
↑	R	R	R	
≥≡≤ ≥≠≤	B	R	R	
¬	B	E	B	
∧	B	B	B	
v	B	B	B	
⊃	B	B	B	
≡	B	B	B	
:=	Svdr	A∧(RvBvW)	dl	M if for (:=)
if	X	X	B	
then	dr	X	SvRvBvL	
else	dr	SvRvBvL	dl	
go to	S	E	L	
for	X	E	X	m → M(:=)
step	X	R	R	m
until	X	X	R	
while	X	R	B	m
do	S	X	SvE	
begin	X	E	SvDvE	m
end	S	X	X	
(	dr	X	X	m
)	dl	X	E	
:	S	A	SvE	m
to	X	R	R	
bool, real, int	D	E	X	m
procedure	D	E	X	m
array	D	E	X	m
switch	D	E	X	m
own	D	E	X	m
value	D	E	X	m
label	D	E	X	m
define	X	E	X	m
where	X	X	X	
to be	S	X	SvE	
;	dr	SvDvE	SvDvE	
,	X	X	X	
comment	X	E	X	

KEY:
X undefined
R real or integer
B boolean
A undeclared atom
L label
S statement
E empty
W switch
m minor modifier
M major modifier
dr dynamic right
dl dynamic left

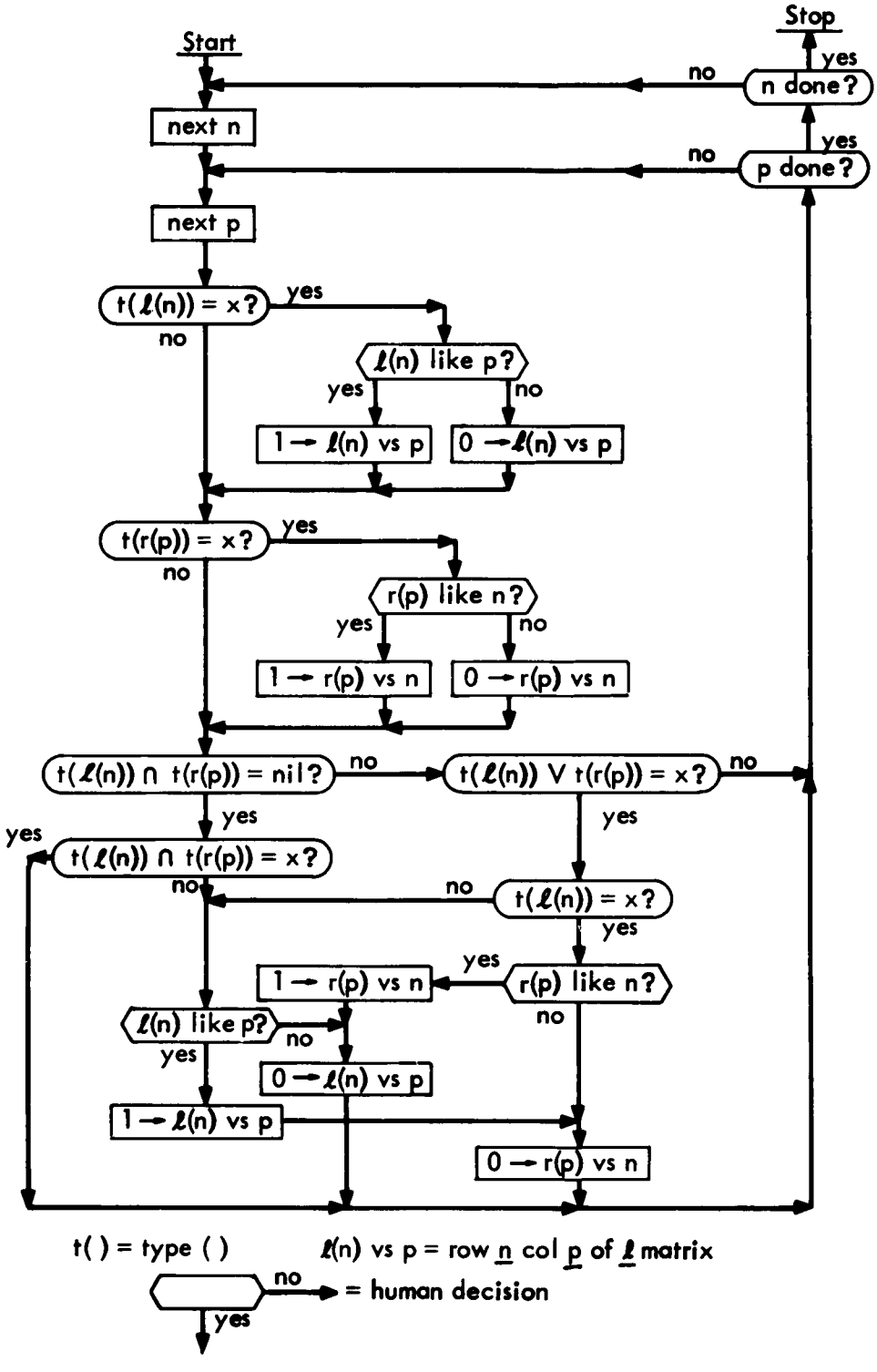
A-1 **Datatypes and Datalikes for ALGOL 60**



A-2 Left Wordlike Matrix for ALGOL 60



A-3 Right Wordlike Matrix for ALGOL 60



A-4 The Like Matrix Setting Algorithm

and matrices for the existing Algol 60 vocabulary to show what a complete and natural language formed from those words should be.

Slight modifications of the Matrix Setting Algorithm can be made to check for inconsistencies in the human responses or to permit ambiguous languages to be defined. The version of Figure A4 accepts the most recent human decision unquestioningly, and also guarantees that the help which detects ambiguities in the Parsing Algorithm will never be reached.

For Algol 60 we handle the entire question of precedence types and plikes in a cavalier fashion with the following fringe cut, in order to save space. The plike question here is so trivial that Plike Matrices and elaborate discussion of precedence types seem out of place. Using the notation A(B) to show B in the A context we have:

ℓ1(:=) plikes only until, while, for (,)

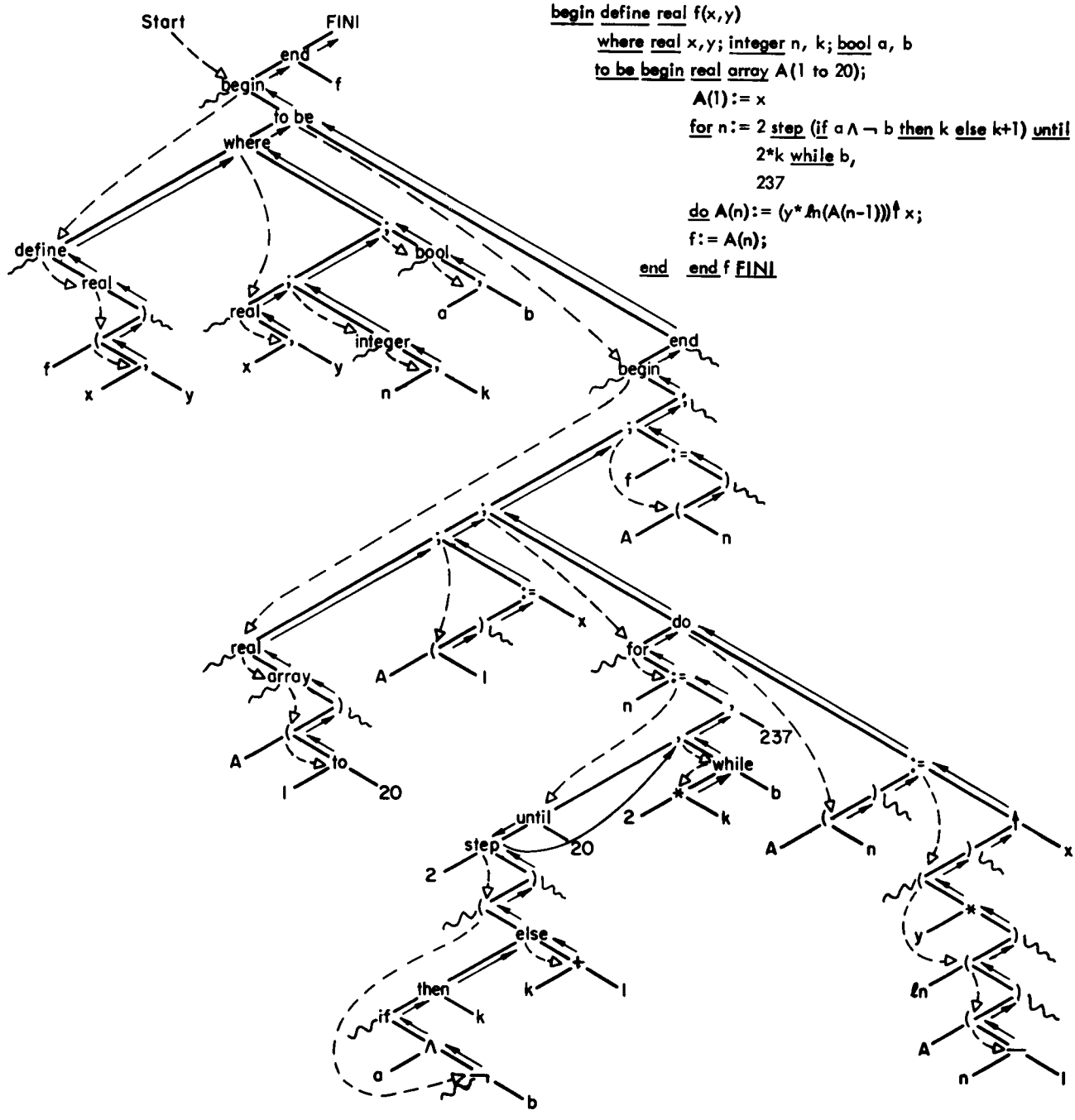
r1(step) plikes only for (,), ;

r1(until) plikes only step

r1(while) plikes only for (,), for (:=)

In Algol 60, everything else plikes everything else.

Finally, Figure A5 shows a non-trivial (but nonsensical) grammatical Algol 60 program in input string and first-pass structure form as generated by the algorithm of Figure 15.




```
begin define real f(x,y)
    where real x,y; integer n, k; bool a, b
    to be begin real array A(1 to 20);
            A(1) := x
            for n := 2 step (if a ∧ ¬ b then k else k+1) until 20,
                    2*k while b,
                    237
            do A(n) := (y* ln(A(n-1)))↑ x;
            f := A(n);
    end   end f FINI
```

A-5 An Example in ALGOL 60 Language

BIBLIOGRAPHY

1. Gorn, S., Reports on the Common Programming Language Task, The Moore School of Electrical Engineering, University of Pennsylvania, 1960.

2. Oettinger, A. G., Automatic Syntactic Analysis and the Pushdown Store, Proc. Symposia in Applied Mathematics, AMS, 1961, pp 104-128.

3. Ross, D. T., A Generalized Technique for Symbol Manipulation and Numerical Calculation, Comm. ACM, Vol. 4, No. 3., March 1961, pp 147-150.

4. Samelson, K. and Bauer, F. L., Sequential Formula Translation, Comm. ACM, Vol. 3, No. 2, February 1960, pp 76-83.

5. Sherry, M. E., Syntactic Analysis in Automatic Translation, Ph. D. Thesis, Harvard University, January 1961, 346 pp.

6. Rhodes, I., The NBS Method of Syntactic Integration, Proc. National Symposium on Machine Translation, Los Angeles, February 1960.

7. Markov, A. A., Theory of Algorithms, Academy of Sciences, USSR, 1954. (Trans. by Jacques J. Schorr-Kon, et al, Offices of Technical Services, Washington), 444 pp.

8. Huskey, H. D. and Wattenburg, W. H., A Basic Compiler for Arithmetic Expressions, Comm. ACM, Vol. 4, No. 1, January 1961, pp 3-9.

9. Floyd, R. W., On Syntactic Analysis and Operator Precedence, CA-62-2, Computer Associates, Inc., August 1962, 31 pp.

10. Irons, E. T., A Syntax Directed Compiler for ALGOL 60, Comm. ACM, Vol. 4, No. 1, January 1961, pp. 51-55.

11. Chomsky, N., On the Notion "Rule of Grammar", Proc. Symposia in Applied Mathematics, AMS, 1960, pp 6-24.

12. Naur, P. (Ed.), Report on the Algorithmic Language ALGOL 60, Comm. ACM, Vol. 3, No. 5, May 1960, pp 299-314.

13. Yngve, V. H., The Depth Hypothesis, Proc. Symposia in Applied Mathematics, AMS, 1960, pp 130-138.

14. Warshall, S., A Syntax Directed Generator, Proc. 1961 EJCC, AFIPS Vol. 20, December 1961, pp 295-305.

15. Ceccato, S., et al, Entire Issue, Methodos, Vol. 12, Nos. 45, 46, 47, 1960.

BIBLIOGRAPHY (Continued)

16. Symposium on the Structure of Language and its Mathematical Aspects Entire Issue, AMS, 1961.

17. Comm. ACM, Vol. 4, No. 3, 1961, Entire Issue.

18. Proc. of National Symposium on Machine Translation, Los Angeles, 1960.

INITIAL DISTRIBUTION LIST FOR
CONTRACT AF-33(600)-42859
DSR 8753 2/63

GOVERNMENT AND MILITARY

Air Force Mem. Prod. Equipment
Attn: Mr. Robert W. Phillips
Redistribution Group
Room 2D823 Pentagon
Washington 25, D.C.

AMC Ballistic Missiles Center
Attn: LBPR (Mr. F. Becker)
Air Force Unit Post Office
Los Angeles 45, California

Army Materiel Command Board
Attn: Col. D. MacFeeters
Aberdeen Proving Grounds
Maryland

ASTIA (10)
Arlington Hall Station
Arlington 12, Virginia

Commander
Aeronautical Systems Division
Attn: Mr. W.M. Webster ASRCTF
Wright-Patterson Air Force Base
Ohio (2 and letter)

Commander
U.S.N.O.T.S.
Pasadena Annex
Attn: J.H. Jennison
3202 E. Foothill Boulevard
Pasadena, California

Commanding Officer
Ordnance Materials Res. Office
Watertown Arsenal
Attn: N.L. Reed, Asst. Director
Watertown 72, Massachusetts

Director
Naval Research Laboratories
Attn: Code 2021
Washington 25, D.C.

U.S. Atomic Energy Commission
Technical Information Service
P.O. Box 62
Oak Ridge, Tennessee

INDUSTRIAL AND UNIVERSITY

Aerojet-General Corporation
Attn: Myra Grenier, Librarian
1100 W. Hollyvale
Azusa, California

Aerojet-General Corporation
Attn: Technical Library 2410-2015A
P.O. Box 1947
Sacramento 9, California

Aeronca Manufacturing Corporation
Engineering Library
Engineering Department
Middletown, Ohio

Aeronutronic
Division Ford Motor Company
Library
Ford Road
Newport Beach, California

Aerospace Industries Association
Attn: Library
1725 DeSales Street, NW
Washington 6, D.C.

Aerospace Industries Association
Technical Service Library
Attn: J.A. Maurice
7660 Beverly Boulevard
Los Angeles, California

Airborne Instruments Laboratory
Attn: Miss Nancy Pannier
Walt Whitman Road
Melville, L.I., New York

Allison Division GMC
Attn: Engineering Library, Plant 8
P.O. Box 894
Indianapolis 6, Indiana

Aluminum Company of America
Attn: D.I. Alkire, Project Engineer
2210 Harvard Avenue
Cleveland 5, Ohio

American Machine and Foundry
Research and Development Division
Attn: Library
689 Hope Street
Stamford, Connecticut

A.O. Smith Corporation
Attn: Technical Library
P.O. Box 584
Milwaukee 1, Wisconsin

Armour Research Foundation
Attn: Dr. S. Hori
Senior Research Engineer
10 West 35th Street
Chicago 16, Illinois

Arthur D. Little, Incorporated
Attn: Miss Dorothy E. Hart
Library (Rt. to David N. Smith)
Acorn Park
Cambridge 40, Massachusetts

Autonetics
Division of North American Aviation, Inc.
9150 E. Imperial Highway, 3041-13, Bldg. 60
Attn: R.L. Panek, Technical Library
Downey, California

AVCO Corporation
Lycoming Division
Attn: H. Maehl
Superintendent Manufacturing Engineering
Stratford, Connecticut

AVCO Corporation
Lycoming Division
Attn: Dr. Hans Klein
Chief, Gas Turbine
Computing Department
Stratford, Connecticut

Battelle Memorial Institute
505 King Avenue
Attn: Mr. Charles E. Day
Report Library
Columbus 1, Ohio

Beech Aircraft Company
Attn: R. H. Owen
Mgr. Manufacturing Engineering
Wichita, Kansas

Bell Aircraft Corporation
Attn: J.C. Millikin
Manager Production Engineering
Internal Zone C-53
Buffalo 5, New York

Bendix Corporation
Attn: Central Library
Department 75
Box 1159
Kansas City 41, Missouri

The Bendix Corporation
Industrial Controls Section
Attn: Technical Library
21820 Wyoming
Detroit 37, Michigan

The Bendix Corporation
Attn: Mr. G.S. Knapf
21820 Wyoming
Detroit 37, Michigan

The Bendix Corporation
Attn: Reports Library
Research Laboratories Division
Southfield, Michigan

Boeing Airplane Company
Vertol Division
Attn: Technical Librarian
Morton, Pennsylvania

The Boeing Company
MASD-Wichita Branch
Attn: 7100-Library-11
3801 South Oliver
Wichita, Kansas

Boeing Airplane Company
Attn: H.E. Laughlin
Orgn. 2-3930 M.S. 45-33
P.O. Box 3985
Seattle 24, Washington

The Boeing Company
Attn: G.M. Fair, Orgn. 2-3010
Numerical Control Representative
P.O. Box 3707 M.S. 12-61
Seattle 24, Washington

The Boeing Company
Attn: E.F. Carlberg
Orgn. 2-5321
Applied Math M.S. 59-81
P.O. Box 3707
Seattle 24, Washington

Bolt Beranek and Newman Incorporated
Attn: Library
50 Moulton Street
Cambridge 38, Massachusetts

Burroughs Corporation
Electro-Data M. and E. Division
Attn: Library
460 Sierra Madre Villa
Pasadena, California

Carnegie Institute of Technology
Attn: Professor Allen Newell
Systems and Communications Sciences
Schenley Park
Pittsburgh 13, Pennsylvania

Carnegie Institute of Technology
Attn: Technical Library
Schenley Park
Pittsburgh 13, Pennsylvania

CEIR Incorporated
Attn: R. L. Trexler
621 Farmington Avenue
Hartford, Connecticut

CEIR Incorporated
Attn: Library
1200 Jefferson-Davis Highway
Arlington 2, Virginia

Cessna Aircraft Company
5800 East Pawnee
Attn: Engineering Library
Wichita, Kansas

Chance Vought Library
Unit 1-63101
Rt.-Campbell, Schuyler, Schwind
P.O. Box 5907
Dallas 22, Texas

Cincinnati Milling Machine Company
Attn: Mrs. Hamilton
Engineering Library
Oakley Cincinnati 9, Ohio

Cincinnati Milling Machine Company
Attn: Dr. Eugene Merchant
4701 Marburg Avenue
Cincinnati 9, Ohio

Cleveland Pneumatic Tool Company
Attn: Engineering Library
3761 E. 77th Street
Cleveland 5, Ohio

Concord Control Incorporated
Attn: Mr. J.O. McDonough, Pres.
Boston 35, Massachusetts

Continental Aviation and Engineering Corp.
Attn: Technical Library
12700 Kercheval Avenue
Detroit 15, Michigan

Control Engineering
Attn: R.J. Barber
330 West 42nd Street
New York 36, New York

Convair Plant I
Division of General Dynamics Corporation
Attn: Mr. M.D. Weisinger
Chief of Applied Mfg. Res. and Product Dev.
Pacific Highway
San Diego 12, California

Curtiss-Wright Corporation
Wright Aeronautical Division
Attn: H. H. Downs
Engineering Library
Wood-Ridge, New Jersey

Douglas Aircraft Company, Incorporated
Aircraft Division
Attn: Technical Library
3855 Lakewood Boulevard
Long Beach, California

Douglas Aircraft Company, Incorporated
Attn: N.H. Shappell, Manager Mfg.
Missiles and Space Division
Santa Monica, California

Douglas Aircraft Company, Incorporated
Missile and Space Systems Library
Dept. A2-260
Santa Monica, California

Ex-Cell-O Corporation
Attn: John F. Garon
Numera-Trol Sales
P.O. Box 386
Detroit 32, Michigan

Farrand Optical Company, Incorporated
Attn: Library
4401 Bronx Boulevard
New York 70, New York

Ferranti Electric Incorporated
Attn: Mr. R.H. Davies
Plainview, L.I., New York

Ford Motor Company
Attn: L. Ording, Mgr. Mfg. Plant
Eng. Dept. Trans. and Chassis Division
36200 Plymouth Road
Livonia, Michigan

Franklin Institute
Attn: Miss Marian Johnson
Technical Report Library
20th and Parkway
Philadelphia 3, Pennsylvania

General Dynamics-Astronautics
Digital Computing Laboratory
Attn: H. W. Buckner
Mail Zone 101-70
P.O. Box 1128
San Diego 12, California

General Dynamics -Fort Worth Division
Attn: B. J. McWhorter
Dept. 6, Box 011
Aerosystems Computation Laboratory
P.O. Box 748
Fort Worth, Texas

General Electric Company
Manager-Advanced Manufacturing Eng.
LJED-Manufacturing Op., Mail Code E-122
Building 700, Evendale Plant
Cincinnati 15, Ohio

General Electric Company
Attn: W. W. Spencer, Manager
Process Control Engineering
570 Lexington Avenue
New York 22, New York

General Electric Company
Specialty Control Department
Library (Rt.-Manager-Program Control Sales)
Waynesboro, Virginia

General Machine Company
Attn: Thomas Hebeln
Technical Library
3628 West Pierce Street
Milwaukee 15, Wisconsin

General Motors Research Labs
Data Processing Department
Attn: Librarian
12 Mile and Mound Roads
Warren, Michigan

Giddings and Lewis Machine Tool Company
Attn: Mr. H. E. Ankeney
Fond Du Lac, Wisconsin

Goodyear
Eng. Adm. and Planning
Attn: Librarian
1210 Massillon Road
Akron 15, Ohio

Grumman Aircraft Eng. Corporation
Engineering Library, Plant 5
Bethpage, L.I., New York

Grumman Aircraft Engineering Corporation
Attn: G. D. Fogel
Automatic Computing Group
Plant 5
Bethpage, L. I., New York

Grumman Aircraft Engineering Corporation
Attn: Angelo Galgano
Manufacturing Engineering, Plant 3
Bethpage, New York

Hiller Aircraft Corporation
Engineering and Research Library
1350 Willow Road
Palo Alto, California

Hughes Aircraft Company
P.O. Box 11337, Emery Park Station
Attn: Plant Library, Bldg. 1
Tucson, Arizona

Hughes Tool Company
Attn: William W. Lampkin
V.P. Mfg. Aircraft Division
Florence Avenue and Teale Street
Culver City, California

Hydo-Mill Company
Attn: Mr. Harry Emrich V.P.
1707 Cloverfield Boulevard
Santa Monica, California

IBM
Attn: S. Matsa
Math and Applications Dept.
1271 Avenue of Americas
New York 20, New York

IBM Liaison Office
Room 26-147
77 Massachusetts Avenue
Cambridge 39, Massachusetts

IBM Data Processing Division
Attn: Mr. F. E. Chappelear
P.O. Box 4014
Beverly Hills, California

Itek Corporation
Attn: Mr. Norman H. Taylor, V.P.
10 Maguire Road
Lexington 73, Massachusetts

Jones and Lamson Machine Company
Attn: N. R. Heald, Mgr. Development
Springfield, Vermont

Kaiser Aircraft Electric Corporation
Attn: J.B. Olson
Chief Engineer
P.O. Box 11275 Station A
Palo Alto, California

Kearney and Trecker Corporation
Attn: Engineering Technical Library
11000 Theodore Trecker Way
Milwaukee 14, Wisconsin

KPT Manufacturing Company
Engineering Library
Locust Avenue
Roseland, New Jersey

Ladish Company
Attn: Metallurgical Library
5481 S. Packard Avenue
Cudahy, Wisconsin

Lockheed Aircraft Company
Attn: Dr. L.H. Ferrish
Org. 7901, Building 103
Sunnyvale, California

Lockheed-California Company
Attn: Central Library
Dept. 72-25, Building 63
P.O. Box 551
Burbank, California

Lockheed Aircraft Corporation
Attn: Robert Vaughn
Producibility Methods Eng.
2555 N. Hollywood Way
Burbank, California

Lockheed Aircraft Corporation
Science-Technology Information Center
Dept. 72-34, Zone 26
Marietta, Georgia

McDonnell Aircraft Corporation
McDonnell Automation Center
Dept. 73
P.O. Box 516
St. Louis 66, Missouri

McDonnell Aircraft Corporation
Attn: Engineering Library
Department 218
P.O. Box 516
St. Louis 66, Missouri

The Marquardt Corporation
Attn: W.E. Otto, Program Manager
Box 670
Ogden, Utah

The Marquardt Corporation
Attn: Engineering Library
16555 Saticoy Street
Van Nuys, California

The Marquardt Corporation
Attn: E. C. Krueske
Mail Zone 21-24
16555 Saticoy Street
Van Nuys, California

The Martin Company (2)
Research Library, A-52
P.O. Box 179 (Rt. to F.L. Blassingame)
Denver 1, Colorado

National Machine Tool Bld. Assn.
Attn: Thomas E. Lloydy
2139 Wisconsin Avenue
Washington 7, D.C.

North American Aviation, Incorporated
Attn: Technical Library
International Airport
Los Angeles 9, California

North American Aviation, Incorporated
Attn: Whitson C. Walter D/187-030
International Airport
Los Angeles 9, California

North American Aviation, Incorporated
Attn: O. Dale Smith, D/187-030
International Airport
Los Angeles 9, California

North American Aviation, Incorporated
Attn: Mr. Robert G. Heckathorne
Numerical Sciences Group 282-072
International Airport
Los Angeles 9, California

North American Aviation, Incorporated
Attn: Technical Information Center
4300 E. Fifth Avenue
Columbus 16, Ohio

Northrop Corporation
Norair Division
Attn: Technical Information, 3924-31
1001 E. Broadway
Hawthorne, California

Nortronics-Systems Support
Attn: Gordon Wilcox, Librarian
500 E. Orangethorpe Avenue
Anaheim, California

Onsrud Machine Works, Incorporated
Attn: Mr. Earle Pnkonin
V.P. Director of Engineering
7700 North Lehigh Avenue
Chicago 31, Illinois

Philco Corporation
C and E Division
Attn: Mrs. C. Ferguson, Librarian
4700 Wissahickon Avenue
Philadelphia 44, Pennsylvania

Republic Aviation Corporation
Attn: Engineering Library
Mr. R.E. Fidoten/Schneider
Farmingdale, L.I., New York

Robert A. Keyes Association
Attn: Mr. Robert A. Keyes
821 Franklin Avenue
Garden City, L.I., New York

Rocketdyne
A Division of North American Aviation, Inc.
Attn: Library, Department 586-306
6633 Canoga Avenue
Canoga Park, California

Rocketdyne
A Division of North American Aviation, Inc.
Solid Rocket Division
Attn: Library
McGregor, Texas

Rohr Corporation
Attn: D.L.S. McCoy
P.O. Box 878
Chula Vista, California

Ryan Aeronautical Company
Attn: Robert L. Clark
Vice President Manufacturing
2701 Harbor Drive
San Diego 12, California

Sanders Associates, Incorporated
Attn: Technical Library
95 Canal Street
Nashua, New Hampshire

Sandia Corporation
Livermore Laboratory
Attn: Technical Library
P.O. Box 969
Livermore, California

The Service Bureau Corporation
Technical Library
635 Madison Avenue
New York 22, New York

Sikorsky Aircraft Division
United Aircraft Corporation
Attn: Library
North Main Street
Stratford, Connecticut

Solar Aircraft Company
Attn: J.A. Logan
Mfg. Factory Division
San Diego 12, California

Sperry Gyroscope Company
Attn: Engineering Library
Mail Station 1A38
Great Neck, New York

Sperry Rand Univac
Numerical Control
Attn: Gastone Chingari
2520 West Sixth Street
Los Angeles 57, California

Stanford Research Institute
Attn: Engineering Library
Menlo Park
California

Stanford Research Institute
Attn: Mr. P.D. Tilton
Industrial Research Engineer
820 Mission Street
South Pasadena, California

Studebaker Corporation
Attn: Mr. C.E. Gierke
Manager-Manufacturing Engineering
635 South Main Street
South Bend 27, Indiana

Sundstrand Machine Tool
Attn: Gordon Nordstrom
Director of Engineering
Belvidere, Illinois

Systems Development Corporation
Technical Library Services
Rt. to A. Rosenberg, C. Kellog
2500 Colorado Avenue
Santa Monica, California

Temco Electronics
Attn: Library, Dept. 403
P.O. Box 6118
Dallas 22, Texas

Thompson Ramo Wooldridge
Attn: Librarian
8433 Fallbrook Avenue
Canoga Park, California

Thompson Ramo Wooldridge
Industrial Control Systems
Attn: J. J. Childs, Mgr. of ICS Sales
455 Sheridan Avenue
Michigan City, Indiana

Thompson Ramo Wooldridge Incorporated
Attn: K.C. White
Staff Director-Industrial Engineering
23555 Euclid Avenue
Cleveland 17 Ohio

Union Carbide Nuclear Company
ORGDP Central Library
Attn: J.L. Gabbard, Jr.
P.O. Box P
Oak Ridge, Tennessee

United Aircraft Corporation
Research Laboratories Library
East Hartford 8, Connecticut

United Aircraft Corporation
Research Laboratories
Attn: C. Robinson
East Hartford, Connecticut

United States Rubber Company
Research and Dev. Dept., Library
Alps Road
Wayne, New Jersey

Univac Division Remington Rand
Attn: Mr. W.R. Lonergan
Systems Programming
351 Park Avenue South
New York 10, New York

Univac, Division Sperry Rand Corporation
Attn: Mr. R.W. Bemer
351 Park Avenue South
New York 10, New York

University of California
Department of Engineering
Attn: Dr. Allen B. Rosenstein
405 Hilgard Avenue
Los Angeles 24, California

The Warner and Swasey Research Center
Attn: Mr. S.F. Winchell,
Director of Research and Development
28999 Aurora Road
Cleveland 39, Ohio

The Warner and Swasey Company
Attn: Technical Library
5701 Carnegie Avenue
Cleveland 3, Ohio

Westinghouse Electric Corporation
Division Engineering Library
Steam Division
Lester, Pennsylvania

Westinghouse Electric Corporation
Technical Information Center
P.O. Box 1693
Baltimore 3, Maryland

Wyman-Gordon Company
Grafton Plant
Technical Information Center
Worcester 1, Massachusetts

AD Electronic Systems Laboratory Massachusetts Institute of Technology Cambridge 39, Massachusetts AN ALGORITHMIC THEORY OF LANGUAGE by Douglas T. Ross. November, 1962. 68p. incl. illus. (Contract No. AF-33(600)-42859). Report ESL-TM-156. Unclassified report The Algorithmic Theory of Language takes the view that processing algorithms define classes of language. A language belongs to a class depending upon whether or not it is properly processed by the corresponding algorithm. Following preliminary statement of n-component element and plex definitions, several General Principles concerning the step-by-step growth of large, complex structures are introduced. The words and symbols of language are then considered to be elements with attractive and repulsive properties which cause them to link together to form linguistic structures. The General Principles are applied to suitable element definitions to yield (over)	UNCLASSIFIED UNCLASSIFIED

AD Electronic Systems Laboratory Massachusetts Institute of Technology Cambridge 39, Massachusetts AN ALGORITHMIC THEORY OF LANGUAGE by Douglas T. Ross. November, 1962. 68p. incl. illus. (Contract No. AF-33(600)-42859). Report ESL-TM-156. Unclassified report The Algorithmic Theory of Language takes the view that processing algorithms define classes of language. A language belongs to a class depending upon whether or not it is properly processed by the corresponding algorithm. Following preliminary statement of n-component element and plex definitions, several General Principles concerning the step-by-step growth of large, complex structures are introduced. The words and symbols of language are then considered to be elements with attractive and repulsive properties which cause them to link together to form linguistic structures. The General Principles are applied to suitable element definitions to yield (over)	UNCLASSIFIED UNCLASSIFIED

AD Electronic Systems Laboratory Massachusetts Institute of Technology Cambridge 39, Massachusetts AN ALGORITHMIC THEORY OF LANGUAGE by Douglas T. Ross. November, 1962. 68p. incl. illus. (Contract No. AF-33(600)-42859). Report ESL-TM-156. Unclassified report The Algorithmic Theory of Language takes the view that processing algorithms define classes of language. A language belongs to a class depending upon whether or not it is properly processed by the corresponding algorithm. Following preliminary statement of n-component element and plex definitions, several General Principles concerning the step-by-step growth of large, complex structures are introduced. The words and symbols of language are then considered to be elements with attractive and repulsive properties which cause them to link together to form linguistic structures. The General Principles are applied to suitable element definitions to yield (over)	UNCLASSIFIED UNCLASSIFIED

AD Electronic Systems Laboratory Massachusetts Institute of Technology Cambridge 39, Massachusetts AN ALGORITHMIC THEORY OF LANGUAGE by Douglas T. Ross. November, 1962. 68p. incl. illus. (Contract No. AF-33(600)-42859). Report ESL-TM-156. Unclassified report The Algorithmic Theory of Language takes the view that processing algorithms define classes of language. A language belongs to a class depending upon whether or not it is properly processed by the corresponding algorithm. Following preliminary statement of n-component element and plex definitions, several General Principles concerning the step-by-step growth of large, complex structures are introduced. The words and symbols of language are then considered to be elements with attractive and repulsive properties which cause them to link together to form linguistic structures. The General Principles are applied to suitable element definitions to yield (over)	UNCLASSIFIED UNCLASSIFIED

AD

derivations of successively more elaborate algorithms defining the behavior of these elements, and generating in one left-to-right pass the First-Pass Structure which explicitly exhibits the syntactic and semantic structure of a statement by showing syntactic context by a tree structure and semantic context by the "precedence string". The present development stops with the concepts of major and minor modifiers and leaves ambiguity resolution and other topics to future papers.

UNCLASSIFIED

UNCLASSIFIED

AD

derivations of successively more elaborate algorithms defining the behavior of these elements, and generating in one left-to-right pass the First-Pass Structure which explicitly exhibits the syntactic and semantic structure of a statement by showing syntactic context by a tree structure and semantic context by the "precedence string". The present development stops with the concepts of major and minor modifiers and leaves ambiguity resolution and other topics to future papers.

UNCLASSIFIED

UNCLASSIFIED

AD

derivations of successively more elaborate algorithms defining the behavior of these elements, and generating in one left-to-right pass the First-Pass Structure which explicitly exhibits the syntactic and semantic structure of a statement by showing syntactic context by a tree structure and semantic context by the "precedence string". The present development stops with the concepts of major and minor modifiers and leaves ambiguity resolution and other topics to future papers.

UNCLASSIFIED

UNCLASSIFIED

AD

derivations of successively more elaborate algorithms defining the behavior of these elements, and generating in one left-to-right pass the First-Pass Structure which explicitly exhibits the syntactic and semantic structure of a statement by showing syntactic context by a tree structure and semantic context by the "precedence string". The present development stops with the concepts of major and minor modifiers and leaves ambiguity resolution and other topics to future papers.

UNCLASSIFIED

UNCLASSIFIEI

AD
Electronic Systems Laboratory
Massachusetts Institute of Technology
Cambridge 39, Massachusetts
AN ALGORITHMIC THEORY OF LANGUAGE
by Douglas T. Ross. November, 1962.
68p. incl. illus. (Contract No/ AF-33(600)-42859). Report ESL-TM-156.
Unclassified report

UNCLASSIFIED

The Algorithmic Theory of Language takes the view that processing algorith s define classes of language. A language belongs to a class depending upon whether or not it is properly processed by the corresponding algorithm. Following preliminary statement of n-component element and plex definitions, several General Principles concerning the step-by-step growth of large, complex structures are introduced. The words and symbols of language are then considered to be elements with attractive and repulsive properties which cause them to link together to form linguistic structures. The General Principles are applied to suitable element definitions to yield
(over)

UNCLASSIFIED

AD
Electronic Systems Laboratory
Massachusetts Institute of Technology
Cambridge 39, Massachusetts
AN ALGORITHMIC THEORY OF LANGUAGE
by Douglas T. Ross. November, 1962.
68p. incl. illus. (Contract No. AF-33(600)-42859). Report ESL-TM-156.
Unclassified report

UNCLASSIFIED

The Algorithmic Theory of Language takes the view that processing algorithms define classes of language. A language belongs to a class depending upon whether or not it is properly processed by the corresponding algorithm. Following preliminary statement of n-component element and plex definitions, several General Principles concerning the step-by-step growth of large, complex structures are introduced. The words and symbols of language are then considered to be elements with attractive and repulsive properties which cause them to link together to form linguistic structures. The General Principles are applied to suitable element definitions to yield
(over)

UNCLASSIFIED

AD
Electronic Systems Laboratory
Massachusetts Institute of Technology
Cambridge 39, Massachusetts
AN ALGORITHMIC THEORY OF LANGUAGE
by Douglas T. Ross. November, 1962.
68p. incl. illus. (Contract No. AF-33(600)-42859). Report ESL-TM-156.
Unclassified report

UNCLASSIFIED

The Algorithmic Theory of Language takes the view that processing algorithms define classes of language. A language belongs to a class depending upon whether or not it is properly processed by the corresponding algorithm. Following preliminary statement of n-component element and plex definitions, several General Principles concerning the step-by-step growth of large, complex structures are introduced. The words and symbols of language are then considered to be elements with attractive and repulsive properties which cause them to link together to form linguistic structures. The General Principles are applied to suitable element definitions to yield
(over)

UNCLASSIFIED

AD
Electronic Systems Laboratory
Massachusetts Institute of Technology
Cambridge 39, Massachusetts
AN ALGORITHMIC THEORY OF LANGUAGE
by Douglas T. Ross. November, 1962.
68p. incl. illus. (Contract No. AF-33(600)-42859). Report ESL-TM-156.
Unclassified report

UNCLASSIFIED

The Algorithmic Theory of Language takes the view that processing algorithms define classes of language. A language belongs to a class depending upon whether or not it is properly processed by the corresponding algorithm. Following preliminary statement of n-component element and plex definitions, several General Principles concerning the step-by-step growth of large, complex structures are introduced. The words and symbols of language are then considered to be elements with attractive and repulsive properties which cause them to link together to form linguistic structures. The General Principles are applied to suitable element definitions to yield
(over)

UNCLASSIFIED

AD

derivations of successively more elaborate algorithms defining the behavior of these elements, and generating in one left-to-right pass the First-Pass Structure which explicitly exhibits the syntactic and semantic structure of a statement by showing syntactic context by a tree structure and semantic context by the "precedence string". The present development stops with the concepts of major and minor modifiers and leaves ambiguity resolution and other topics to future papers.

UNCLASSIFIED

UNCLASSIFIED

AD

derivations of successively more elaborate algorithms defining the behavior of these elements, and generating in one left-to-right pass the First-Pass Structure which explicitly exhibits the syntactic and semantic structure of a statement by showing syntactic context by a tree structure and semantic context by the "precedence string". The present development stops with the concepts of major and minor modifiers and leaves ambiguity resolution and other topics to future papers.

UNCLASSIFIED

UNCLASSIFIED

AD

derivations of successively more elaborate algorithms defining the behavior of these elements, and generating in one left-to-right pass the First-Pass Structure which explicitly exhibits the syntactic and semantic structure of a statement by showing syntactic context by a tree structure and semantic context by the "precedence string". The present development stops with the concepts of major and minor modifiers and leaves ambiguity resolution and other topics to future papers.

UNCLASSIFIED

UNCLASSIFIED

AD

derivations of successively more elaborate algorithms defining the behavior of these elements, and generating in one left-to-right pass the First-Pass Structure which explicitly exhibits the syntactic and semantic structure of a statement by showing syntactic context by a tree structure and semantic context by the "precedence string". The present development stops with the concepts of major and minor modifiers and leaves ambiguity resolution and other topics to future papers.

UNCLASSIFIED

UNCLASSIFIED

REPORTS PUBLISHED

"Papers on the APT Language," Douglas T. Ross and Clarence G. Feldmann, Technical Memorandum 8436-TM-1, Electronic Systems Laboratory, June 1960.

"Method for Computer Visualization," Albert F. Smith, Technical Memorandum 8436-TM-2, Electronic Systems Laboratory, September 1960.

"A Digital Computer Representation of the Linear, Constant-Parameter Electric Network," Charles S. Meyer, Technical Memorandum 8436-TM-3, Electronic Systems Laboratory, August 1960.

"Computer-Aided Design: A Statement of Objectives," Douglas T. Ross, Technical Memorandum 8436-TM-4, Electronic Systems Laboratory, September 1960.

"Computer-Aided Design Related to the Engineering Design Process," S. A. Coons and R. S. Mann, Technical Memorandum 8436-TM-5, Electronic Systems Laboratory, October 1960.

"Investigations in Computer-Aided Design, Interim Report No. 1," Project Staff, Report 8436-IR-1 for period December 1, 1959 to May 30, 1960.

"Automatic Feedrate Setting in Numerically Controlled Contour Milling," J. D. Welch, Report 8436-R-1, Electronic Systems Laboratory, December 1960.

"Investigations in Computer-Aided Design, Interim Report No. 2," Douglas T. Ross, Steven A. Coons, et al, Report 8436-IR-2 for period June 1, 1960 to February 28, 1961.

"Design of a Remote Display Console", Glenn C. Randa, Report ESL-R-132, Electronic Systems Laboratory, February 1962.

"Investigations in Computer-Aided Design for Numerically Controlled Production, Interim Report No. 3 and 4," Douglas T. Ross, Steven A. Coons, et al, Report ESL-IR-138 , for period March 1, 1961 to February 28, 1962.